NEARDATA

Funded by the European Union

HORIZON EUROPE FRAMEWORK PROGRAMME

NEARDATA

(grant agreement No 101092644)

Extreme Near-Data Processing Platform

D2.1 Initial Architecture Specifications

Due date of deliverable: 30-06-2023
Actual submission date: 30-06-2023

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	92
WP/Task related to this document	WP2 / T2.1
WP/Task responsible	Universitat Rovira i Virgili (URV)
Leader	Xavier Roca (URV)
Technical Manager	Josep Lluís Berral (BSC), Raúl Gracia (DELL) and Pedro García (URV)
Quality Manager	André Martin (TUD) and Daniel Barcelona (URV)
Author(s)	Pedro García (URV), Xavier Roca (URV), Josep Lluís Berral (BSC), Aaron Call (BSC), Gonzalo Gomez (BSC), Lorena Alonso (BSC), André Martin (TUD), Zhaoyu Chen (NCT), Max Kirchner (NCT), Theodore Alexandrov (EMBL), Maciej Malawski (SANO), Jan Przybyszewski (SANO), Enrique Chirivella-Perez (KIO), Daniel Ramirez (KIO), André Miguel (SCO), Raúl Gracia (DELL) and Paolo Ribeca (UKHS)
Partner(s) Contributing	URV, BSC, TUD, NCT, EMBL, SANO, KIO, SCO, DELL, UKHS
Document ID	NEARDATA_D2.1_Public.pdf
Abstract	Deliverable D2.1 "Initial Architecture Specifications" aims to present in detail the initial specifications of the overall architecture, the description of the use cases, the experiments and the benchmarking frameworks that we proposed in the NEARDATA project. Specifically, this deliverable contains the identification of the requirements and the description of the main components and their integration with the different use cases. On the other hand, the use cases will be presented together with the necessary modifications to adapt and integrate them to the described components. Next, the different benchmarking frameworks and their requirements will be provided, together with the test environments to perform the experiments. Finally, ethical and confidentiality measures have been established to prevent and secure the treatment of extremely sensitive data.
Keywords	Architecture, components, confidential computing, use cases, experiments, extreme data, sensitive data, ethical requirements.

History of changes

Version	Date	Author	Summary of changes
0.1	15-05-2023	Xavier Roca (URV)	First draft.
0.2	18-06-2023	Xavier Roca (URV), Josep Lluís Berral (BSC), Aaron Call (BSC), Gonzalo Gomez (BSC), Lorena Alonso (BSC), André Martin (TUD), Zhaoyu Chen (NCT), Max Kirchner (NCT), Theodore Alexandrov (EMBL), Maciej Malawski (SANO), Jan Przybyszewski (SANO), Enrique Chirivella-Perez (KIO), Daniel Ramirez (KIO), André Miguel (SCO), Raúl Gracia (DELL) and Paolo Ribeca (UKHS)	Contributions.
0.5	21-06-2023	Daniel Barcelona (URV), Pedro García (URV), Xavier Roca (URV), Aaron Call (BSC) and Raúl Gracia (DELL)	Internal review of the deliverable.
1.0	29-06-2023	Xavier Roca (URV)	Final version.

Table of Contents

1 Introduction **3**

2 Initial architecture specifications **5**
2.1 Proposed theoretical architecture . . . 5
2.1.1 Data plane: Data Catalog and Data Connectors . . . 6
2.1.2 Control Plane: Data Broker and Confidential Data Orchestration . . . 7
2.2 Proposed Architecture . . . 8
2.3 NEARDATA Architecture Life Cycle . . . 10
2.4 Data Plane components . . . 13
2.4.1 Lithops: Serverless data processing platform . . . 13
2.4.2 Serverless data connector for dynamic unstructured data partitioning . . . 16
2.4.3 Pravega: A Tiered Storage System for Data Streams . . . 18
2.5 Control Plane components . . . 28
2.5.1 Confidential Technologies . . . 28
2.5.2 Confidential data exchange . . . 32
2.5.3 Confidential orchestration . . . 34
2.5.4 AI-based optimization of Cloud/Edge Workflows . . . 37

3 Description of use case scenarios **41**
3.1 Clinical sequencing of human pathogens . . . 41
3.1.1 Description of the use case . . . 41
3.1.2 Datatypes and datasets . . . 42
3.1.3 Data connectors . . . 42
3.1.4 Experiments . . . 43
3.1.5 Technical challenges . . . 44
3.2 Variants interaction analyses in massive genomics datasets . . . 44
3.2.1 Description of the use case . . . 44
3.2.2 Datatypes and Datasets . . . 46
3.2.3 Data connectors . . . 46
3.2.4 Experiments . . . 46
3.2.5 Technical challenges . . . 47
3.3 Transcriptomics Atlas Use Case . . . 47
3.3.1 Description of the use case . . . 47
3.3.2 Datatypes and Datasets . . . 48
3.3.3 Data connectors . . . 48
3.3.4 Experiments . . . 49
3.3.5 Technical challenges . . . 49
3.4 Metabolomics Use Case . . . 50
3.4.1 Description of the use case . . . 50
3.4.2 Datatypes and Datasets . . . 51
3.4.3 Data connectors . . . 51
3.4.4 Experiments . . . 52
3.4.5 Technical challenges . . . 53
3.5 Surgery Use Case . . . 54
3.5.1 Description of the use case . . . 54
3.5.2 Datatypes und Datasets . . . 54
3.5.3 Data Connectors . . . 54
3.5.4 Experiments . . . 55
3.5.5 Technical Challenges . . . 55

4 Benchmarking framework **56**
4.1 Clinical sequencing of human pathogens . . . 56
4.1.1 Involved Tools and Systems . . . 57
4.2 Variants Interaction Analytics in massive genomics datasets . . . 57
4.2.1 Cloud and edge computing environments . . . 57
4.2.2 Pipeline evaluation . . . 58
4.2.3 Involved Tools and Systems . . . 58
4.3 Transcriptomics Atlas Use Case . . . 59
4.3.1 Atlas Pipeline . . . 59
4.3.2 Involved Tools and Systems . . . 60
4.3.3 Federated Learning Pipeline . . . 61
4.4 Metabolomics Use Case . . . 62
4.4.1 METASPACE pipeline . . . 62
4.4.2 Experiment 1 . . . 62
4.4.3 Experiment 2 . . . 63
4.4.4 Involved Tools and Systems . . . 63
4.4.5 Evaluation and impact . . . 63
4.5 Surgery Use Case . . . 64
4.5.1 Surgical Navigation Pipeline . . . 64
4.5.2 Federated Learning Pipeline for Surgical Data . . . 65
4.5.3 Involved Tools and Systems . . . 66
4.6 Testbed . . . 66
4.6.1 KIO Networks Cloud Computing Architecture (VDC) . . . 67
4.6.2 KIO Networks Cloud Computing GPU Architecture (VDC-GPU) . . . 68
4.6.3 KIO Networks Cloud Computing S3 Architecture (S3) . . . 68
4.6.4 KIO Networks Cloud Computing Kubernetes Architecture . . . 69
4.6.5 Partners requirements . . . 70

5 Confidential and ethical requirements for AI technologies **72**
5.1 Clinical Sequencing of Human Pathogens Use Case . . . 73
5.2 Variants Interaction Use Case . . . 74
5.3 Transcriptomics Atlas Use Case . . . 75
5.4 Metabolomics Use Case . . . 75
5.5 Confidentiality and ethical considerations in Surgery Use Case . . . 76
5.6 Summary and Contingency of Ethical Issues . . . 77

6 Conclusions **79**

A Appendix: NEARDATA APIs **80**
A.1 Data Plane APIs . . . 80
A.1.1 Data Catalog APIs . . . 80
A.1.2 Data Connectors APIs . . . 80
A.1.3 Streaming APIs . . . 83
A.2 Control Plane APIs . . . 87
A.2.1 SCONE CAS API . . . 88
A.2.2 Keycloak API . . . 88

List of Abbreviations and Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
AWS	Amazon Web Services
BSC	Barcelona Supercomputing Center
CAS	Configuration and Attestation Service
CGS	Computer-Guided Surgery
CPU	Central Processing Unit
CSV	Comma-separated values
DAG	Directed Acyclic Graph
DMZ	Demilitarized Zone
DNA	Deoxyribonucleic Acid
DSAD	Dresden Surgical Anatomy Dataset
EMBL	European Molecular Biology Laboratory
ENA	European Nucleotide Archive
ETL	Extract, Transform and Load
FaaS	Function as a Service
FDR	False Discovery Rate
FUSION	Finland-United States Investigation of NIDDM Genetics
GDPR	General Data Protection Regulation
GENEVA	Gene Environment Association Studies initiative
GERA	Genetic Epidemiology Research on Aging
GPU	Graphics Processing Unit
GWAS	Genome-Wide Association Studies
HDFS	Hadoop Distributed File System
HPC	High Performance Computing
IAM	Identity and Access Management
IoT	Internet of Things
IPC	Inter-Process Communication
LTS	Long Term Storage
ML	Machine Learning

MS	Mass Spectrometry
NCBI	National Center for Biotechnology Information
NCT	German Cancer Research Center
NFS	Network File System
NHS	National Health Service
NIDDM	Non-Insulin-Dependent Diabetes
NUgene	Northwestern University NUgene project
OLTP	Online Transactional Processing
OS	Operating System
PII	Personally Identifiable Information
REST	Representational State Transfer
RNA	Ribonucleic Acid
SANO	Centre for Computational Medicine
SCO	Scontain
SEV	Secure Encrypted Virtualization
SLA	Service Level Agreements
SLAM	Simultaneous Localisation and Mapping
SRA	Short Read Archive
SSD	Solid State Drive
TEE	Trusted Execution Environment
TUD	Technical University Dresden
UKHS	UK Health Security Agency
URV	Universitat Rovira i Virgili
UTM	Unified Threat Management
VDC	Virtual Data Center
VM	Virtual Machine
WAL	Write-Ahead Log
WTCCC	Welcome Trust Case Control Consortium

1 Introduction

The NEARDATA project is based on the idea of near-data processing paradigm. We can describe this concept as the ability to place processing power close to the data, rather than sending the data to the processor.

The goal of NEARDATA is to create an extreme near-data infrastructure mediating data flows between Object Storage and Data Analytics platforms across the Compute Continuum. Our novel Data Plane is an intermediary data service that intercepts and optimises data flows (S3 API, stream APIs) with high performance near-data connectors (Cloud/Edge). Our unique Data Broker service will provide secure data access and orchestration of dispersed data sources thanks to TEEs and federated learning architectures. Our NEARDATA platform is a novel technology for data mining of large and dispersed unstructured data sets that can be deployed in the Cloud and in the Edge (HPC, IoT Devices), that leverages advanced AI technologies and offers a novel confidential cybersecurity layer for trusted data computation.

From this definition three core objectives can be distinguished:

- *Provide high-performance near-data processing for Extreme Data Types*: The first objective is to create a novel intermediary data service (Data Plane) 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.

- *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.

- *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 and integrity, 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.

Validating a project like NEARDATA and demonstrating its impact requires a consistent evaluation from the main components that form the overall architecture along robust use cases and benchmarks. To this end, we focus on three health data domains containing large unstructured data: metabolomics (images), genomics (text), and surgery data (video). The selected data domains comply perfectly with the definition of extreme data.

OMICs settings have a relevant problem of huge and increasing data volumes that push current technologies to their limits. Here, there is a strong challenge in the data ingestion from Object Storage to computing analytics services. As we will see, the ingest-then-compute data-shipping paradigm is

becoming unfeasible for many OMICs datasets. In this line, moving terabytes of compressed information to the memory of analytics clusters just for pre-processing and data manipulation can become extremely expensive. This is a huge problem inside the Cloud, but it gets even worse if data must be moved to remote locations (Edge/HPC).

On the other hand, computer-assisted surgery shows an extreme problem of data speed because it requires low latency real time video analytics and robotic IoT event streams. There is here a strong challenge to support real-time video streams that must be processed very fast to the Object Storage at large scale. Furthermore, low-latency requires here near-data processing at the Edge to satisfy real-time processing requirements.

This document provides an overview of the project and the initial specifications of the novel NEARDATA platform. In the following sections, the main components of the overall architecture will be described along with their specifications and approaches to integrations with the use cases to meet the described objectives and address the problems caused by the massive data ingestion and low latency real time video analytics. Next, the use cases will be detailed in an extended way to observe their requirements, datasets and pipelines. Likewise, the benchmarking frameworks will be defined for their correct evaluation.

Finally, health data in general is highly sensitive, so it has tough privacy and security requirements that preclude many hospitals and research labs to share, move, or publish their data openly. Ethical and confidentiality requirements will be detailed to guarantee a correct use of the data as well as its protection in the different confidential and federated learning use cases.

2 Initial architecture specifications

2.1 Proposed theoretical architecture

Figure 1 shows the overall architecture proposed in the NEARDATA project. This solution is inspired by the Reference Architecture of International Data Spaces[1].

Our proposal is based on four main blocks:

- **Data Sources**. This block encompasses the different data sources found in the NEARDATA platform. These data sources can be located in different environments, such as Cloud or Edge. Thus, data can be directly stored in storage services such as Object Storage or collected and ingested through different IoT devices.
- **Analytics**. This block is in charge of collecting the different processing platforms to apply the different computing cases shown in Figure 1, such as Federated Computing, Massive Parallel Serverless Computing and High-Performance Computing.
- **Data Plane**. This block is focused on data management within the NEARDATA platform. It provides data indexing, metadata and semantics through the Data Catalog component, connection between Data Processing Platforms with Data Sources offering different Data Connectors and Stream Operators. The Data Plane allows analytic frameworks (Data Pipelines) to ingest extreme data efficiently.
- **Control Plane**. As the name indicates, the purpose of this block is to establish control over the different blocks. It allows the discovery of confidential data through the Data Broker component. Thanks to the Trusted Execution Environments component, the different blocks explained above can be secured to perform confidential execution and data access. Likewise, these secure executions can be orchestrated by the Confidential & Federated Orchestration component. Finally, the Control Plane will incorporate an AI-based component to optimize data pipelines.

In addition, we can identify five key entities:

- **Data Provider** is responsible for making data available for consumption by users. In most cases, the Data Provider will be the Data Source platforms. Data Sources in Figure 1 are Data Providers.
- **Data Consumer** is in charge of retrieving the data from the Data Provider. The Data Consumer can connect only through Data Connectors or directly to Data Sources to consume the data. In Figure 1, the different analytics frameworks are Data Consumers.
- **The Identity Provider** is the element capable of authenticating users to establish secure connections over the Trusted Execution Environment. It is used to control access and manage the roles of users who wish to use Confidential computing and federated learning. The Identity Provider is found as an authentication service within the Data Broker in Figure 1.
- The **Data Broker** is NEARDATA's main confidentiality and security provider. It contains elements to guarantee secure executions (applications to protect components), elements capable of managing and orchestrating user access and roles and provides confidential data discovery and access. Keeping the same naming convention, this Data Broker concept is related to the Trusted Execution Environment and the Data Broker depicted in Figure 1.
- **The Data Connector** allows connections to be established between the Data Provider and the Data Consumer. Within the context of NEARDATA, the Data Connector will add computation near the Data Sources so that the data is prepared before being consumed. We extend this concept also to Stream Operators in Figure 1.

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

The integration of all the blocks is based on simple paths, as shown in the Figure 1. The process starts with use cases presented by users. To carry out the solutions to these problems, Data Pipelines are developed that use Data Programming Abstractions to connect to the different Data Connectors or Stream Operators to consume the data they require for their execution. Data Connectors will use the Data Catalog (obtain the metadata) to understand the data and apply the correct methods to ingest and prepare it for the computation. By adding the Control Plane functions, the developed pipelines can be optimized for a more efficient execution while taking full advantage of the resources offered by the different processing platforms.

On the other hand, users may propose federated computing use cases, where confidentiality is essential. For this, Trusted Execution Environments will be generated and orchestrated to guarantee a private and secure execution. Finally, users will be able to establish a connection with the Data Broker to discover and access encrypted data and start the execution of their TEE.

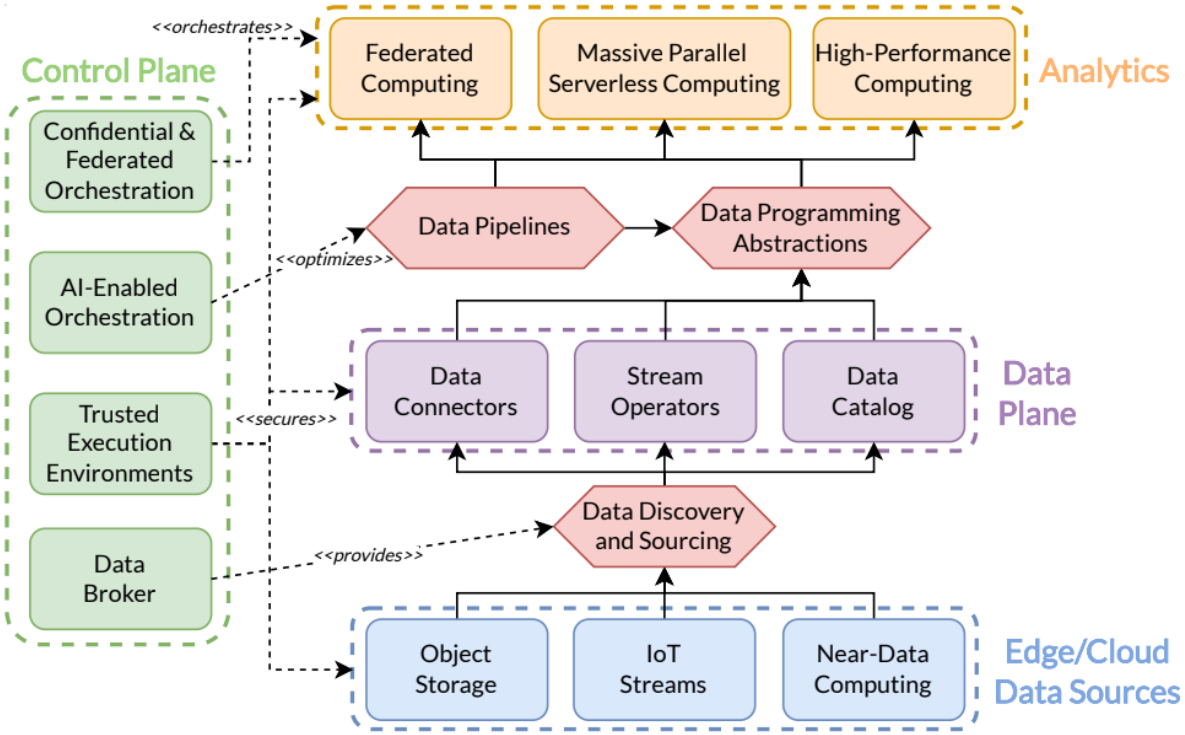


Figure 1: Proposed Theoretical Architecture

2.1.1 Data plane: Data Catalog and Data Connectors

In our platform, Extreme Data Types are virtual objects that encapsulates Unstructured Data, Metadata, and specialised Data Connectors that facilitate its processing and filtering by analytics platforms. Our rich data platform converts bulk unstructured data blobs into semi-structured datasets that can be queried, filtered and parsed.

The Data Plane includes four core components: Data Catalog, Serverless Data Connectors, Stream Operators, and Data Programming abstractions (Analytics interconnection layer).

- The Data Catalog includes an indexing layer on top of Object Storage, in which data is stored as blobs or files in buckets with no semantic information. The Data Catalog layer stores metadata information relevant for data preparation, partitioning, ETL tasks and data query mechanisms. The Data Catalog will be based on Object Storage native metadata services, as well as faster in-memory databases such as Redis.

- The Serverless data connector engine includes an auto-scalable containerised computing service including near-data manipulation primitives that intercepts Object Storage APIs (S3). The connector service leverages the Data Catalog to transform data in an adaptive way and to deliver it to the requesting application with minimal delay and in the appropriate format. Each data type will include a collection of data connectors that will help to provide and transform data to heterogeneous analytics applications running across the compute continuum. Data connectors must also support hardware accelerated substrates not available today in existing serverless Cloud offerings.

- The Stream operator engine provides an auto-scalable containerised computing service including stream data manipulation primitives. The stream connector service must deploy connectors as auto-scalable stream operators offering very fast stateful computations over low-latency event and video streams. To this end, the stream connector service must auto-scale accordingly to the underlying auto-scalable stream storage service.

- Data Programming abstractions: We will implement a Data Analytics interconnection layer offering new data abstractions or integrating existing ones (e.g. Ray Dataset, Spark Dataset, Pytorch Datasets) to facilitate data loading and processing. This key interconnection layer will facilitate an smooth transition for data scientists using widely accepted toolkits. A clear objective here is code transparency for legacy applications and allowing to integrate any analytics (HPC, Edge, Cloud) platform through data and storage wrappers.

2.1.2 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. The Control Plane will be designed as a declarative data analytics interconnection platform enabling a radically-simple, efficient execution of analytics over vast amounts of raw data from heterogeneous sources. This will be done by using the smart and adaptive interconnection of compute and storage data flows thanks to the Data Plane. We will develop a mechanism to translate the declarative specification into a data interconnect, composing user-provided code and data connectors, which will be seamlessly interleaved to deliver heterogeneous connectivity, APIs, and data formats. As a result, this will provide the ability to consume data from anywhere and to run analysis tasks everywhere, without the need for a complex setup or infrastructure expertise.

The Control plane includes four core components: Data Broker, Confidential Compute layer, Confidential and Federated Orchestrator and Data-Driven AI Optimiser.

- The Data Broker is the cockpit of the NEARDATA platform, exposing and orchestrating all confidential services in the Data and Control planes. The Data Broker sources the information from data providers and provides it to data consumers. It permits to create, register and federate Data Sources (Datasets) in a particular domain. It is an extensible platform combining metadata services with data connector services in the Data Plane. Our semantic-rich layer enables rich data discovery and query mechanisms over unstructured or semi-structured data sources located in different repositories. The Data Broker incorporates a service to authenticate the user that will allow the user to access, decrypt and process confidential data. On the other hand, the Data Broker also encompasses the confidential orchestration services and the interconnection of heterogeneous data streams through analytical platforms and services described below.

- The Confidential Compute layer provides protection of data at rest, in transit as well as in use in the data plane thanks to Trusted Execution Environments. We want to ensure confidentiality, integrity and freshness at all times. Confidentiality and integrity can be guaranteed when utilizing state of the art technologies such as Trusted Executions Environments[1] provided through Intel SGX. Although the technology is mature and in the process of being widely adopted, it

is limited to a specific CPU vendor and models like Intel Skylake CPUs. Therefore, an often overlooked requirement is the support of CPU vendors other than Intel such as ARM, AMD and edge devices that provide similar security features to allow service providers to run their applications in an isolated fashion with attestation[2]. We therefore aim at developing a framework that is transparent to the underlying hardware, however, facilitates the available security features provided at the node the application is running on.

- 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 service may leverage Kubernetes orchestration mechanisms but also higher level workflow definitions such as systems like FedML (Federated Learning). The orchestration layer must also provide mechanisms for transparent encryption of data flows, and to leverage privacy-aware data connectors that adapt to data governance policies defined in the Data Catalog. In general, this layer will instrument the different services of the Data Plane to facilitate confidential data interconnection between data providers, data sources and data consumers using a variety of analytic tools across the entire Compute Continuum.

- 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.

2.2 Proposed Architecture

Figure 2 shows the proposed architecture based on the theoretical architecture adopting the real components that we will present below that form the NEARDATA platform.

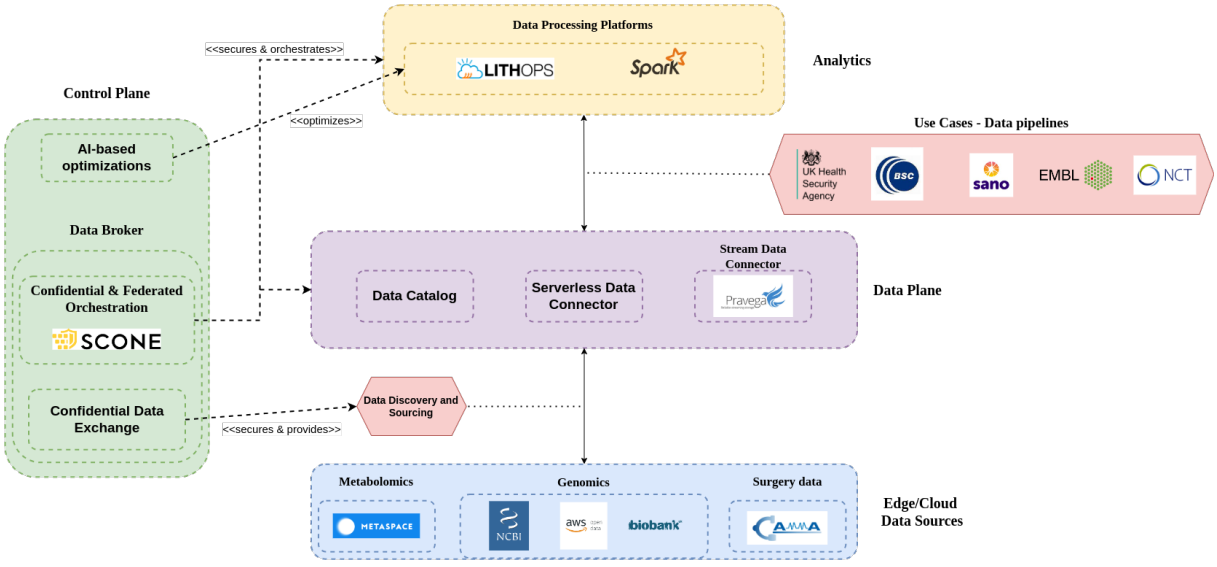


Figure 2: NEARDATA Architecture

Our bottom-up approach first focuses entirely on the specific problems in the selected healthcare domains (OMICS and surgery). In this project, it mainly brings together data scientists from computational medicine and bioinformatics with computational researchers with expertise in Extreme/Big Data and Cloud/Edge Computing. Each partner is focused on a different topic developed in one or more pipelines. We can see the partners presenting use cases in **Use Cases - Data pipelines** block.

UKHS, BSC, SANO and EMBL are focused on OMICS and NTC on surgery videos.

The project focuses mainly on extreme health data to increase the impact of results in this field. This requires extracting large datasets from big data sources. We can distinguish two types of data sources: data stored in the Cloud or in the Edge. Most of the datasets that partners submit for analysis are available on different cloud platforms or in specific data centres. EMBL presents its METASPACE platform that hosts an engine for metabolite annotation of imaging mass spectrometry data, as well as a spatial knowledge base of metabolites from thousands of public datasets provided by the community. METASPACE is considered an extensive Data Source for metabolomics data. On the other hand, different data centres are identified, such as National Center for Biotechnology Information (NCBI) or UK Biobank for genomics data and CAMMA for surgery videos. Extremely sensitive data belonging to an organization or hospital cannot be disseminated and confidentiality must be guaranteed. For this reason, they must be stored privately in the Edge. This is the case for the BSC partner, where all the data they analyze must remain within their organization and the Marenostrum[2]. We can identify some of the different data sources in the block **Edge/Cloud Data Sources**.

As mentioned above, the **Data Plane** facilitates the processing and filtering of extreme data from unstructured data, metadata and specialised data. To meet this objective, four different components are proposed that can be completed through four major projects.

- In our NEARDATA platform we find different Data Sources dedicated specifically to each of the types of health data we present in the project. We can detect that each of these will have its own specialized Data Catalog to extract the necessary metadata to acquire knowledge about the data and facilitate the subsequent tasks of ingestion and processing. A clear example of this is METASPACE, presented as a Data Source for metabolomic data. This platform offers different storage APIs to perform searches on public datasets and collect their metadata. Therefore, METASPACE offers its own Data Catalog. We can see METASPACE definition in section 3.4.

- Lithops is a serverless data analytics platform that is ideally suited for massively parallel data management operations from data in Object Storage, as such, it is an excellent substrate to implement Data Connectors. It offers an extensible storage and compute architecture that supports major Cloud Services in major Cloud providers (IBM Cloud, AWS, Google Cloud, Azure) but also open source on-premise installations (Kubernetes, OpenShift) and Edge platforms (WebAssembly). Lithops also offers an extensible partitioner architecture that permits data-driven parallel processing over Object Storage buckets that leverages the massive aggregate bandwidth of Object Storage. A single parallel map in Lithops chunks data dynamically on-the-fly using the selected partitioner and automatically allocating the required number of parallel functions depending on the data. Lithops already provides data partitioners for different genomics and metabolomics data types.

- The Serverless Data Connector is a data connector that enables to partition data efficiently and dynamically on the fly directly from the object storage. Serverless Data Connector will offer a novel partitioning tool to preprocess scientific data formats. In this way, data consumers (Partners - Use Cases) can take advantage of this framework to facilitate the partitioning and ingestion of large volumes of data.

- Dell Pravega 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. Pravega is a tiered storage system: data events are first durably stored in tier-1, which is a low-latency, durable and replicated write-ahead log. Once data is stored in tier-1, the data is cached in memory for fast real-time access and the client is acknowledged so it can continue writing data. In parallel, Pravega groups small events into larger chunks and writes them to tier-2 or long-term storage

[2] `https://www.bsc.es/es/marenostrum/marenostrum`

(Object Stores, HDFS, etc.). This allows Pravega to achieve a sweet-spot in the latency versus throughput trade-off: writers and real-time processing tasks obtain low latency (thanks to tier-1 and memory cache, respectively), whereas batch jobs get high throughput thanks to parallel reads from long-term storage. Not only that, Pravega adds the notion of "auto-scaling" to the Stream abstraction, which is unique and allows to adapt the degree of parallelism of a Stream automatically based on the load variations.

The **Control Plane** will be designed as a declarative data analytics interconnection platform. The DataBroker will be in charge of establishing a totally secure connection between the different components and the partners to store and retrieve private datasets and perform confidential executions guaranteeing their security. To this end, TUD and SCONTAIN present SCONE, that permits running unmodified programs inside Trusted Execution Environments (TEE). The technologies presented to establish TEEs are CAS (Configuration and Attestation Service) to identify whether an application is trustworthy or not and Keycloak to authenticate users. Finally, BSC presents an AI-based optimization model to guarantee the correct orchestration of resources according to the requirements of the different use cases or data analytics platforms.

All components identified in the proposed architecture will be expanded upon in the following sections. In addition, an appendix has been added with all the APIs offered by each of these components and new ones have been proposed (See Appendix A: NEARDATA APIs) for proper integration and consistency with the life cycles presented below.

2.3 NEARDATA Architecture Life Cycle

In order to develop more clearly the operation and integration of all the components within the NEARDATA platform, different life cycles will be detailed.

First of all, we will identify the most basic execution that we can find within the NEARADATA platform. This will consist of a generic use case for analyzing massive data from a Data Source. To do this, the user will select a data processing platform which will connect to a Data connector to consume the required dataset through the Data Catalog or by accessing directly from the Data Source. Figure 3 shows the sequence diagram that identifies the whole process.

This generic use case is divided into two parts, the preprocessing stage and the data processing stage. The first one is in charge of using the Data Connector to establish a connection between the Data Processing Platform and the Data Catalog or directly to the Data Source. If the Data Catalog is used, Data Connectors will collect metadata to understand the data and apply the correct method to ingest it from the Data Source. If the Data Source is accessed directly, it is considered that the user already knows exactly the file and where it is located to be processed. Both processes will return the dataset to the Data Connector so that the Data Processing Platform can start consuming the data. The second stage initiates the processing steps established within the use case to analyze and extract all possible value from the data. It should be noted that the Data Processing Platform used will be optimized thanks to the AI-based optimizations component of BSC.

From this basic life cycle we can identify different variants due to the complexity and naturalness of the use cases and data formats used. OMICS data are considered extreme data due to their large volumes. Trying to ingest them immediately is practically impossible with current technologies. Therefore, massive job parallelization is necessary to cope with this problem. The dynamic partitioning offered by the Serverless Data Connector component is absolutely paramount when trying to deal with this type of problem. This Data connector will establish two types of connections on the Data Source. The first one to collect all possible metadata to generate the partitions and the second one to extract these dynamically as they are needed in the processing stages.

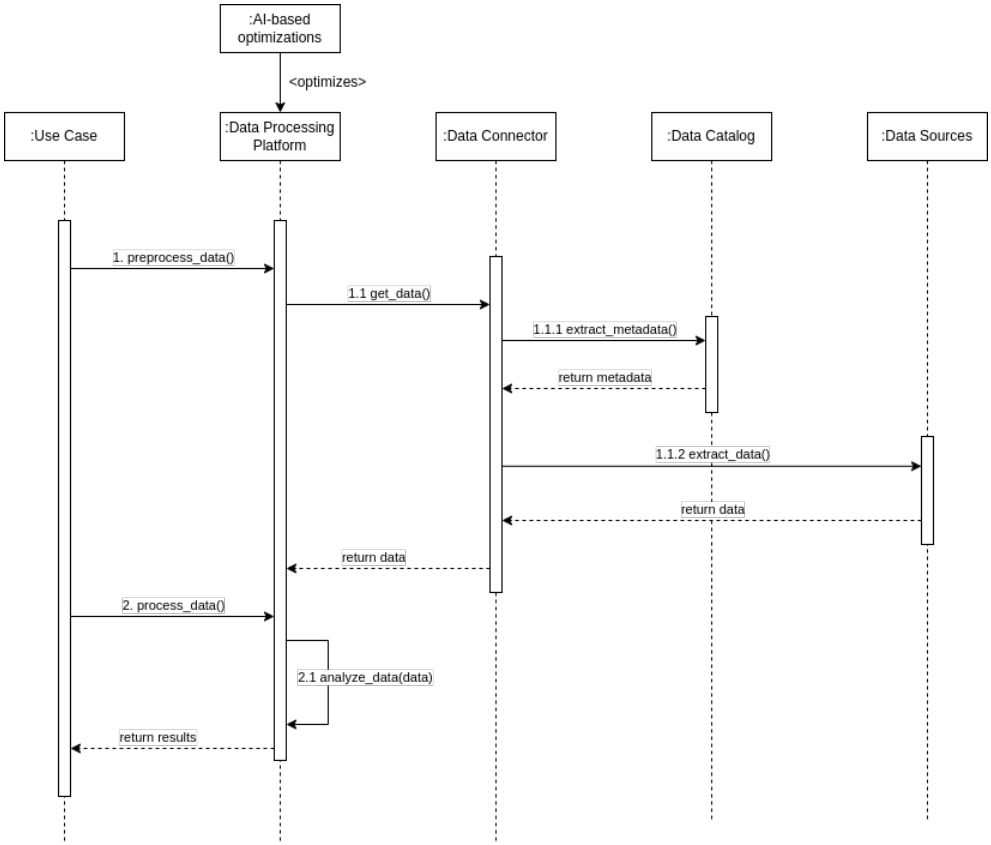


Figure 3: Basic NEARDATA Life Cycle

If we talk about data in video format as in the case of surgery videos. It is key to establish a constant data flow between the Data Processing Platform and the Data Source to be able to analyze the data in real time. It is also necessary to guarantee the minimum latency during this process. In this case, we would speak of a cyclic pattern between data access and data ingestion and analysis. The Pravega Stream Data Connector offered by our platform is essential to address this problem.

Next, we will present the life cycle of a use case based on confidential computing or federated learning that incorporates the basic components to secure the pipeline and authenticate the users capable of executing them. We will highlight a new key element, the Identity Provider, capable of authenticating and managing user access to the configured TEEs. Figure 4 represents a sequence diagram of the life cycle of a confidential use case on a TEE.

First of all, the use case must be secured with the tools offered by the SCONE platform to establish a secure execution environment involving the pipeline, the Data Processing Platform and the Data Connector. In order to access this confidential use case, users will need to have certain keys or certificates capable of proving that they are a trusted user with execution roles on the pipeline. In addition, within a secure environment it is extremely necessary that the sensitive data used is not shared or extracted from the processing platform without preventative measures such as data encryption. This data can only be decrypted by all trusted users who have keys or certificates that give them authority over the encrypted data. Likewise, Data Sources must accept encrypted data and maintain policies to ensure the anonymity, confidentiality and security of the data.

Once a fully secure and configured TEE is available, the user will be able to start the pipeline execution confidentially. The user must be authenticated through the Identity Provider. This key element

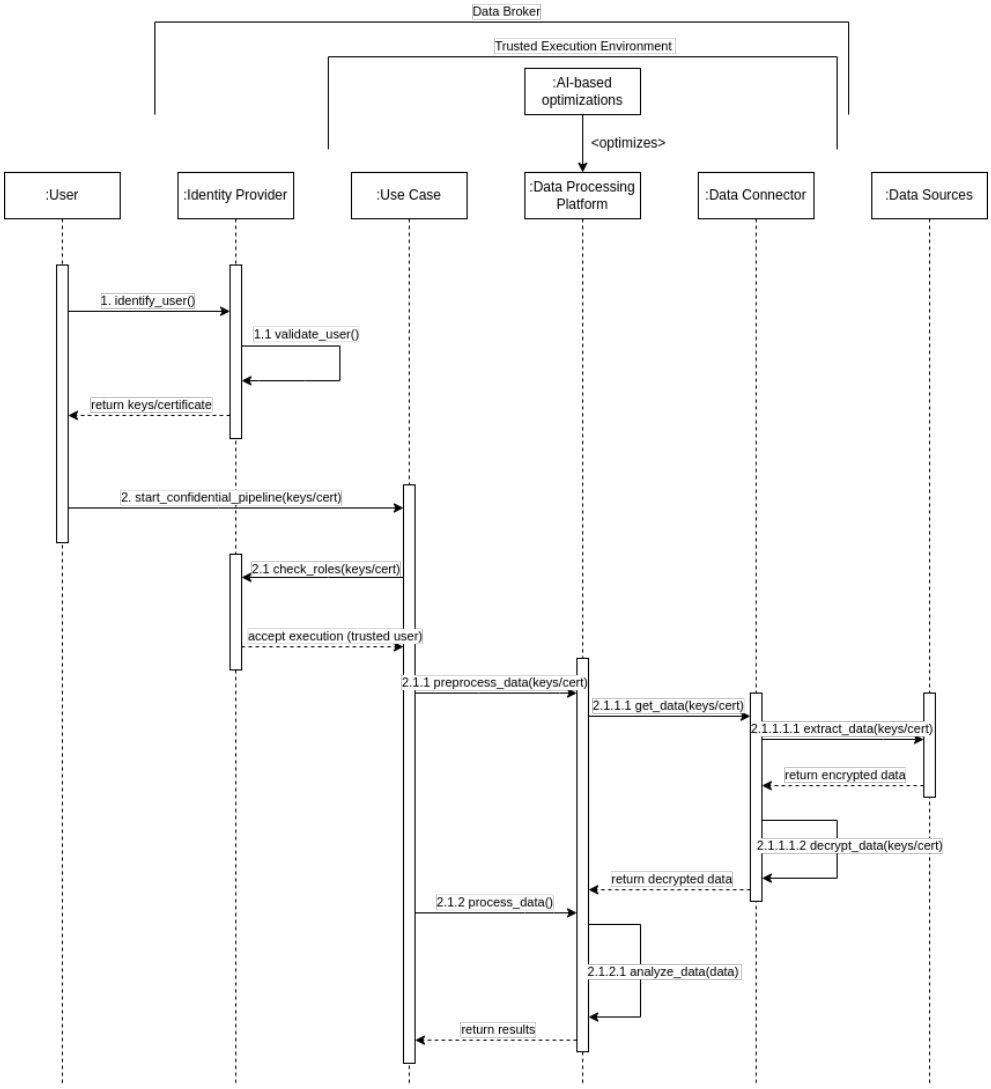


Figure 4: Confidential Computing Life Cycle

will help us to control and manage user identification and access to the TEE. For this, the Keycloak technology presented by SCO offers the necessary services to meet the established requirements. Once the user is authenticated, he/she will try to access the TEE through the keys or certificates returned by the Identity Provider. The TEE will check the user's role in the environment. If the user has the execution role, the user will be able to perform the necessary tasks to extract and analyze the data in a secure and confidential way. If not, the user will not be allowed to execute the use case. The following steps are similar to the previous life cycle of a basic use case. In this case, we can find an important difference when extracting the data. As we have already mentioned, these are encrypted in the Data Source and Data Calatog and we will only be able to access them if the user has the keys or certificates provided by the Identity Provider. Once this data is ingested into the Data Processing Platform, the user will be able to decrypt it with the keys or certificates. Once the data is completely clean, the data processing or analysis stage will begin. As mentioned above, it must be ensured that the data is not extracted from the TEE without being secured through different encryption methods to ensure confidentiality and privacy.

After reviewing the different life cycles that we can find within the proposed architecture we will detail in an extended way each of the components that form the Data Plane and the Control Plane along with its integration within the NEARDATA context and architecture.

2.4 Data Plane components

2.4.1 Lithops: Serverless data processing platform

Lithops [3] is a Python multi-cloud serverless computing framework. It allows to run unmodified local Python code at massive scale on the main serverless computing platforms. Lithops delivers the user's code into the cloud without requiring knowledge of how it is deployed and run. Moreover, its multicloud-agnostic architecture ensures portability across cloud providers, overcoming vendor lock-in.

Lithops provides great value for data-intensive applications like Big Data analytics and embarrassingly parallel jobs. It is specially suited for highly-parallel programs with little or no need for communication between processes.

Also, facilitates consuming data from object storage (like AWS S3, GCP Storage or IBM Cloud Object Storage) by providing automatic partitioning and data discovery for common data formats like CSV. Lithops abstracts away the underlying cloud-specific APIs for accessing storage and provides an intuitive and easy to use interface to process high volumes of data.

Architecture and implementation. The high-level architecture of Lithops is depicted in Figure 5. In its most fundamental incarnation, Lithops leverages just two different cloud services: the compute backend to launch MapReduce jobs; and the storage backend to store all data, including intermediate results. To keep Lithops completely serverless, the compute backend is typically a FaaS platform (e.g., AWS Lambda) and the storage backend is a BaaS storage service (e.g., AWS S3), so that its two main pillars can scale independently from each other.

Internally, the main components of Lithops are:

- *Executor*, which allows end users to execute their code in the cloud through simple API calls. Upon an API call, it serializes and uploads the single-machine user code and input data from a local machine (e.g., laptop) to the storage backend. When a cloud function finishes its execution, the output data generated by executing the user code within the cloud function is persisted to the storage backend. For this reason, the executor monitors the storage backend for the ouput data and transfers it to the user's local machine when available.

- *Invoker*, which performs the "appropriate" number of function invocations against the compute backend. We say "appropriate" since the number of cloud functions depends on the API call itself. The *invoker* can be run on the cloud to hide the high invocation latency when the Lithops client is very far from the compute backend.

- *Worker* is the workhorse of Lithops. In short, it runs on the compute backend, typically as a cloud function, and its main role is to execute the user code associated with the API call that spinned it up. In essence, it fetches the input data and user code from the storage backend, and executes it, eventually saving the output to the storage backend.

Lithops can be seen as a *dynamic* task orchestrator, which can transparently take a user's code and input data, save them into the storage backend, and execute it at large scale by dynamically invoking a *generic worker function*. This approach removes the overhead for function registration, favors the reuse of the single registered function in order to mitigate cold starts, and allows to run user code that exceeds the deployment package size constraints, since dependencies are injected at runtime, not statically.

The *runtime* is the environment where the user code runs. As of today, all the compute backends supported by Lithops permit to run functions in containerized environments. Leveraging Docker technology allows developers to create their own custom runtimes. Simply put, a user may prepare a Docker image with the required packages (Python modules, system libraries and binaries), and then store it to a container registry (e.g., Docker Hub registry). The compute backend service will get the container from the registry the first time is needed. From thereon, the image is cached on a local registry to speed up subsequent invocations. To avoid side effects, both the client and server must

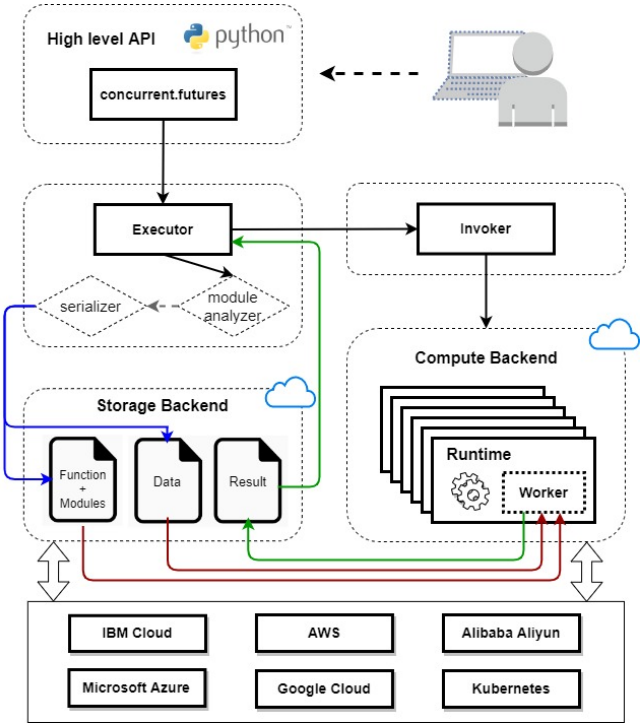


Figure 5: Lithops Architecture

use the same version of Python. To fulfill this need with zero user overhead, Lithops automatically detects the Python version of the client and accordingly deploys the runtime based on this information.

Programming Model. One fundamental principle behind Lithops is programming simplicity. The objective was to make this tool as much usable as possible, irrespective of whether the programmer is a cloud expert or not. As mentioned above, Function Executor is the core object in Lithops. This object allows to perform calls to the Lithops API to run parallel tasks. When an instance of the executor is created, a unique ID is assigned to the instance. This unique ID is used later to keep track of function invocations and the results stored in the storage backend. The executor loads the configuration (e.g., account details) required to grant Lithops access to the compute and storage backends necessary to launch Lithops.

Lithops is shipped with two different high-level Compute APIs, and two high-level Storage APIs.

Compute APIs. Users can perform calls using this API to run parallel tasks that will be executed in the computation backend selected. The first Compute API is the Futures API, which is based on the Python concurrent.futures library [4]. It includes methods to run user code in the cloud, track function executions and download the final results from the storage backend. In addition, the user can create automated plots of the execution trace of the different function invocations.

The second compute API is the *Multiprocessing API*. Lithops implements Python's `multiprocessing` API [5] to *transparently* run local-parallel applications but using serverless functions for Processes and a Redis instance for shared state and Inter-Process Communication (IPC). *Processes* and *Pool* are the abstractions used in multiprocessing to parallelize computation. They interact internally with Lithops's Futures API. Lithops also implements all stateful abstractions from Python `multiprocessing`: Queue, Pipe, Shared memory, Event... . Since FaaS lacks mechanisms for function-to-function communication, a Redis database is used.

Storage APIs. The Storage API makes it straightforward to operate the storage backend with calls similar to those of the Python `boto3` library [6]. Lithops allows to create a **Storage** instance and ab-

stract away the backend implementation details. Lithops also incorporates another API, Storage OS API, which mimics the `os` and the built-in function `open` to access Cloud Storage as if it were a local file system.

Data partitioning. To provide an easy-to-use Map and MapReduce execution platform, a key ingredient is a built-in data partitioner. The only action that a user has to do is to supply the list of object keys comprising the dataset. Once data discovery has finished, the data partitioner enters the scene to seamlessly generate the partitions based on a configurable chunk size parameter. If no chunk size is specified, each object will be processed by a single executor. The great advantage of using this data partitioner is that it is done on the fly without the need to create static partitions stored in the storage backend. As partitions are created, they are distributed among the different map functions. In this way, dividing a file into smaller chunks allows the user to take advantage of the parallelism provided by the compute backends to process the data.

Execution Modes. Lithops compute backends can be classified in three different execution modes depending on the backend the user chooses. The execution mode allows the user to decide where and how the functions are executed.

- Localhost. This mode allows users to run functions in their local machine, by using processes. This is the default mode of execution if no configuration is provided.
- Serverless Mode. This mode allows users to run functions by using publicly accessible Serverless compute services, such as IBM Cloud Functions, Amazon Lambda, among others. In this mode of execution, each function invocation equals to a parallel task running in the cloud in an isolated environment.
- Standalone Mode. This mode allows users to run functions by using one or multiple Virtual machines (VM), either in a private cluster or in the cloud. On each VM, functions run using parallel processes like in the Localhost mode.

Multi-cloud support. Lithops provides an extensible backend architecture that is designed to work with different compute and storage services available on Cloud providers and on-premise backends. In this sense, you can code your application in Python and run it unmodified wherever your data is located at. All services supported by Lithops are described in Figure 6.

Figure 6: Compute and Storage services compatible with Lithops

Integration of Lithops with NEARDATA and use-cases. The role of Lithops within NEARDATA is defined as a serverless data processing platform.

- *Integration with object storage and compute backends.* Lithops is implemented on top of both the object storage and the compute backends offering an abstraction layer and wrappers that allow users to easily run local code on any cloud platform offered by the multiple providers of this service. NEARDATA use cases will benefit from Lithops' generalizations, abstractions and APIs to establish communication between object storage and the different compute backends

to perform their analytics tasks and experiments. Lithops incorporates some benchmarks and experiments where it demonstrates the efficiency of both object storage calls and the invocation of multiple functions in parallel. Users will be able to check which Cloud offers better performance in order to select the one that best suits their needs.

- *Integration with use cases.* BSC and SANO intend to adapt their use cases to use Lithops as a Serverless framework. BSC presents a solution completely based on HPC technologies, therefore, the objective to pursue is based on the modification and adaptation of the pipeline to be able to be executed in the cloud through Lithops. This means exploring parallelization opportunities to take full advantage of functions as a service. Figure 7 depicts Lithops integration diagram with the BSC use case.

  SANO presents a cloud solution based on the processing of transcriptomics data in virtual machines. Therefore, they will make small modifications to make the pipeline execution easier through the computation API offered by Lithops. In addition, massively parallelizing jobs offered by FaaS will be explored to significantly reduce execution time and resource usage.

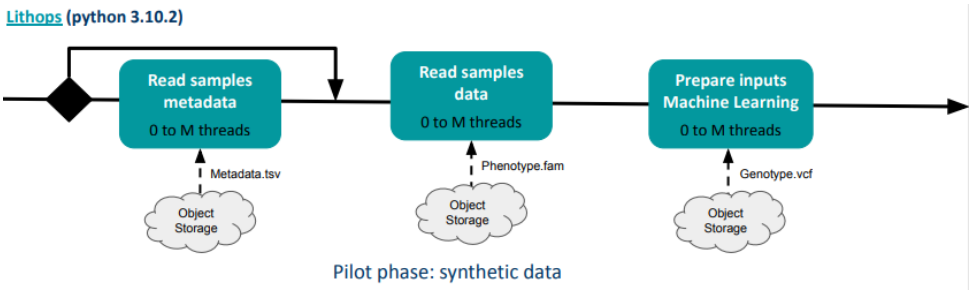


Figure 7: Lithops integration with the BSC use case

2.4.2 Serverless data connector for dynamic unstructured data partitioning

Object storage is the ideal storage for large volumes of data. Incorporating this service in our applications allows us to benefit from the great advantages it offers in terms of bandwidth, availability and durability. It is for this reason that object storage has become the largest source of scientific data in the cloud. There are different cloud providers that directly offer fully accessible and public repositories such as Amazon Web Services Registry of Open Data [7].

The increase in the volume of data that we find today has made the tasks of processing and analysis of these difficult, causing adaptations and modifications to design fully parallelizable architectures to reduce complexity, costs and execution times. To this end, data preprocessing has become an essential task when consuming data. Within the preprocessing stage we can introduce the data partitioning process. In most cases, static partitioning is the most commonly used method for preprocessing large volumes of unstructured data. Static partitioning is based on defining the size or number of chunks to be generated from the beginning of the process. The preprocessing algorithm will use this data to generate the partitions and store them back into the storage system. This type of method is not susceptible to change, i.e., when scientists wish to modify the size or number of chunks they must repeat the entire process from start to finish. Therefore, in workflows where the size of partitions is continuously modified, applying the static partitioning method is not recommended.

To overcome static partitioning and its issues, we devise a novel Serverless Data Connector for dynamic partitioning of unstructured data. Serverless Data Connector will allow the user to partition unstructured data dynamically and in parallel. When we introduce the concept of dynamic we mean that partitions are susceptible to immediate size changes without the need to redo the computation required to create them. Within the NEARDATA context, this Serverless Data Connector establishes a direct connection between the object storage and the data processing platforms allowing the user

to consume his data directly from the object storage in an efficient and parallel way.

Serverless Data Connector is based on the creation of *Generic Partitioning Objects*. These objects have certain tools to preprocess the data and extract all the necessary characteristics to be able to perform the partitioning as needed for each data processing job. The user defines for each type of data who will define how the partitions are generated, being able to choose between different methods such as selecting the number of chunks or establishing certain requirements that the chunks must follow, for example, a certain number of lines.

The *Generic Partitioning Object* will be in charge of generating the partitions from the user's indications. Thus establishing secure byte ranges to dynamically access the original file through requests provided by the Storage APIs offered by the different cloud providers.

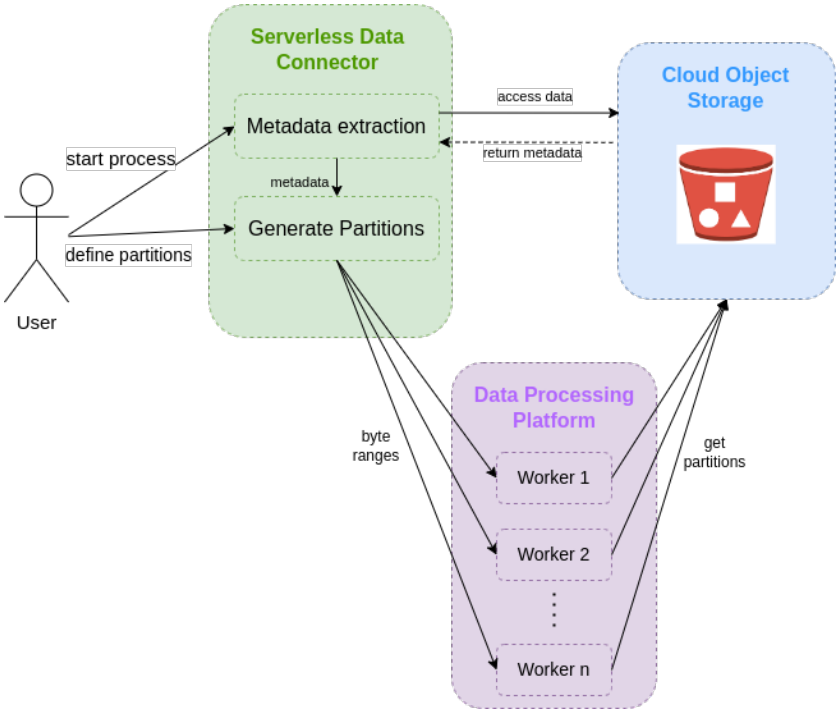


Figure 8: Serverless Data Connector Architecture

Architecture. Figure 8 shows the Serverless Data Connector Architecture. We can distinguish two elementary stages within architecture:

- *Metadata extraction*. In this first phase, the data format will be studied in depth in order to extract those essential elements that represent the data to be partitioned without modifying or corrupting its nature. For this purpose, we will take into account the different patterns that we can find when partitioning an unstructured scientific file.
- *Generate partitions*. To continue with the workflow, users have the opportunity to decide how the data should be partitioned according to their needs. For this purpose, the Partitioning Generic Object must have different methods to suit the user's requirements. Different partitioning methods can be identified, for example, setting a specific number of chunks or specifying a number of lines per chunk. From the metadata extracted from the original file, the Serverless Data Connector will create the different partitions by setting a secure byte range to dynamically access the original file in parallel. Finally, this set of ranges will be distributed in the different parallel workers in order to directly consume the data through requests to the object storage.

The Serverless Data Connector will adapt to the different data processing platforms presented in NEARDATA, offering adaptations to run on backends such as Lithops or Spark according to users' needs.

Integration of the Serverless Data Connector with NEARDATA and use-cases. The role of the Serverless Data Connector within NEARDATA will be defined by the implementation of two of the components specified in the Data Plane, Serverless Data Connector engine and Data Programming abstractions and Wrappers.

- *Integration with the components*. The Serverless Data Connector will be implemented on top of object storage so that it can be defined as a data connector between object storage and distributed parallel processing frameworks. Through its Generic Partitioning Object, it will provide an API for on-the-fly data partitioning that will allow users to consume data in a parallel, efficient and fast manner.
- *Integration with Serverless frameworks*. Different data processing platforms will be able to take advantage of Serverless Data Connector's features. Serverless Data Connector will provide support for Lithops and Spark, the data processing platforms used in NEARDATA.
- *Integration with use-cases*. Serverless Data Connector will present different partitioners for genomics-oriented data format type. Specifically, Serverless Data Connector will be integrated and used for use cases with this data format. By adding this library, use cases will avoid complex preprocessing and partitioning tasks such as the creation of static partitions. This will allow you to take full advantage of the benefits of the Serverless Data Connector to reduce execution times and resource usage.

  The massive genomics analyses use case(s) (Section 3.2) run in a heterogeneous variety of environments (supercomputers, edge nodes, novel architectures such as RISC-V). In addition, the data sets are very large. Consequently, traditional static partitioning forces the user to adapt the size of partitions to each environment. With the help of the serverless data connector, we will have a set of predefined partition sizes and integrate it with a resource manager to decide the appropriate partition size for different environments, thus making it user agnostic. Figure 9 shows how this integration is done through the use case resource manager.

2.4.3 Pravega: A Tiered Storage System for Data Streams

Pravega is an open-source, distributed, tiered storage system for data streams that provides the *streaming fabric that connects to object storage* in NEARDATA. As such, the primary goal for Pravega is to allow applications writing and reading *events*, while durably storing them. Pravega provides users with *client libraries* that implement the Pravega APIs, including the *event writer* and *event reader* ones. Note that applications make sense of events using (de)serializers, as internally Pravega does not keep the notion of events. That is, Pravega manages and stores serialized event data through the IO path.

Pravega stores events in *streams* (like a "topic" in messaging systems). A stream is a durable, elastic, append-only, unbounded sequence of bytes achieving good performance and consistency. Despite events in a stream cannot be modified, there are other valid operations on streams, such as *seal* (i.e., make an stream read-only), *truncate* (i.e., delete data from an arbitrary point in the stream up to its "head"), *scale* (i.e., change in parallelism), and *delete*. Groups of streams are organized into *scopes*, which act as stream namespaces.

Internally, streams are divided into *segments*. A stream segment is a shard or partition of the data within a stream. Similarly to the case of streams, the allowed operations on segments include append, truncate, seal, merge, and delete (but not update). It is worth mentioning that a stream may have multiple segments open for appending events at a given time, which potentially increases parallelism and throughput. In the case of working with streams having parallel segments, applications requiring total order of events are expected to use *routing keys* for writing data. To wit, parallel

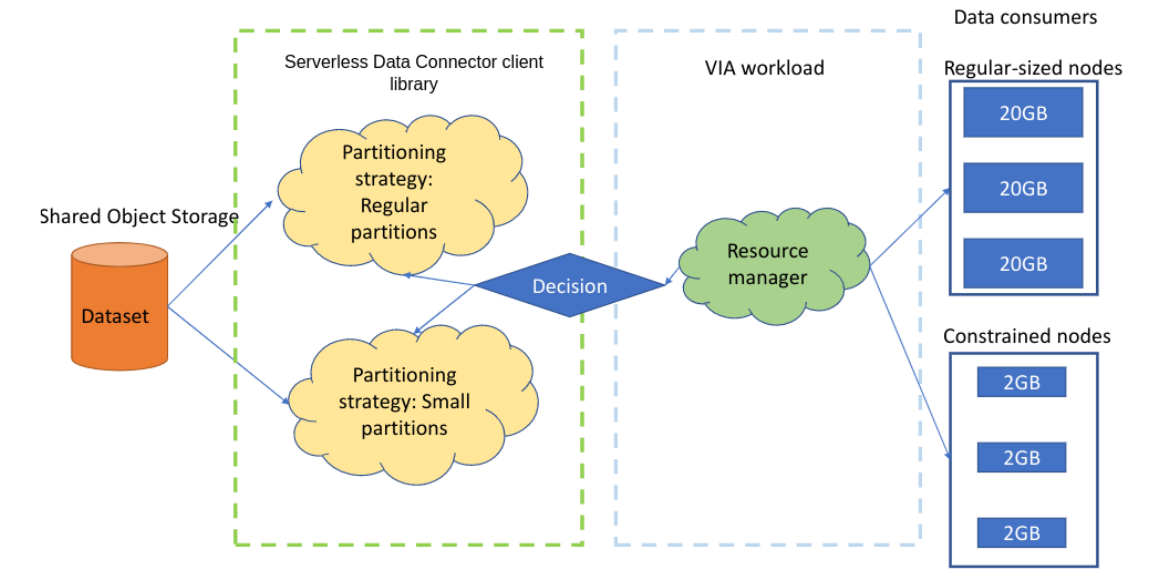


Figure 9: Interaction of VIA with the Serverless Data Connector library, showing how VIA resource manager will interact with Serverless Data Connector library to partition the dataset according to the needs of the consumer nodes (BSC use case).

segments in a stream are assigned to partitions of the key space as a result of a hash function (e.g., $h(k) \in [0,1)$). The writer API can accept as input a user-provided routing key to consistently select the segment for appending events to and, therefore, linearize writes.

An interesting feature of Pravega streams is that they are policy-driven. At the moment, Pravega provides two types of *stream policies*: *retention policies*, which automatically truncate a stream based on size or time bounds; and *auto-scaling policies*, which allow the system to automatically change the segment parallelism of a stream based on the ingestion workload (events/bytes per second). Note that stream policies can be updated along the stream life-cycle.

Architecture. The architecture of Pravega is illustrated in Fig. 10. We identify the following components of the system. First, Pravega writers and readers, which can interact with Pravega server instances either inside the same cluster or from external facilities.

On the server side, we find the Pravega *control plane* formed by one or multiple *controller instances*. The control plane is primarily responsible for orchestrating all stream life-cycle operations, like creating, updating, scaling, and deleting streams. Furthermore, the control plane takes care of enforcing stream policies, including truncating streams according to the retention policy defined and orchestrating the scale up/down of a stream based on the ingestion workload. In the latter case, Pravega builds a feedback loop between the control and data planes, so the control plane can react to the load monitored by the data plane. It is worth mentioning that Pravega can manage a large number of streams. The work associated to managing existing streams is distributed across the available controller instances via a "bucketing" scheme. To wit, streams are associated to a bucket from the number of buckets defined in the system. Buckets are then distributed and owned by controller instances in an attempt to balance stream management load. Moreover, controller instances maintain the stream metadata (which is stored in Pravega itself via the key/value API built on top of streams [8]) and reply to metadata requests about streams.

The *data plane* in Pravega handles data requests from clients and is formed by *segment store instances*. Segment stores play a critical role in making segment data durable and serving it efficiently. Note that segment stores only work with segments and are agnostic to the concept of stream, which is an abstraction built at the control plane. Similarly to the concept of bucket in the control plane, the

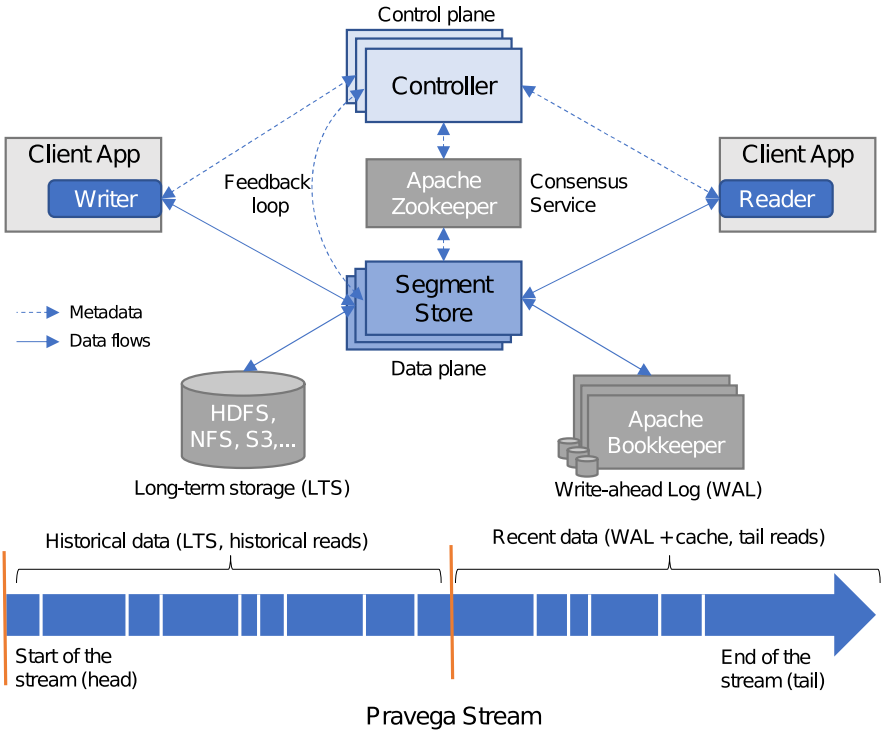


Figure 10: High-level overview of Pravega.

data plane distributes the segment-related load based on *segment containers*. Segment containers are the components that do the heavy lifting on segments and the main role of segment store instances is to host segment containers. A segment is mapped during its entire life to a segment container using a stateless, uniform hash function that is known by the control plane. Thus, "segment ids" belong to a *key-space* that is partitioned across the available segment containers.

The segment store has two primary storage dependencies: *write-ahead log (WAL)* and *long-term storage (LTS)*. The primary goal of WAL is to guarantee that writes are durable with low latency for recovery purposes. Making a write durable means that once the application is acknowledged that the write has succeeded, the system guarantees that the data is written on persistent media and replicated. Currently, Pravega uses Apache Bookkeeper [9, 10] as WAL. Bookkeeper provides excellent write latency for small appends using quorum-based replication.

Segment stores asynchronously migrate data to LTS. Once some data is moved to LTS, the corresponding log file from WAL is truncated. Pravega has a LTS tier for a couple of key assumptions that determined its design: first, data streams are potentially *unbounded* and the system should be able to store a large number of segments in a *cost-effective manner*. Consequently, we need a horizontally-scalable bulk store to accommodate historical stream data, like existing cloud storage services. Second, we need to provide *high throughput and parallelism* for reading *historical data* if applications need to catch up with the stream. In fact, the combination of low-latency real-time writes (WAL) and reads (in-memory cache), plus high throughput historical reads (LTS) allow Pravega to achieve a *sweet spot* in the throughput versus latency trade-off.

Finally, Pravega requires from a consensus service. This is basically needed for leader election and general cluster management purposes. At the moment, Pravega uses Apache Zookeeper [11, 12].

Pravega Streams. Next, we provide an overview of the mechanics involving clients and server-side instances when operating with streams, such as writing and reading. Also, we describe the guarantees that Pravega provides, which are critical to modern streaming applications.

Stream Auto-scaling. Stream auto-scaling is a distinguishing feature of Pravega. It allows to au-

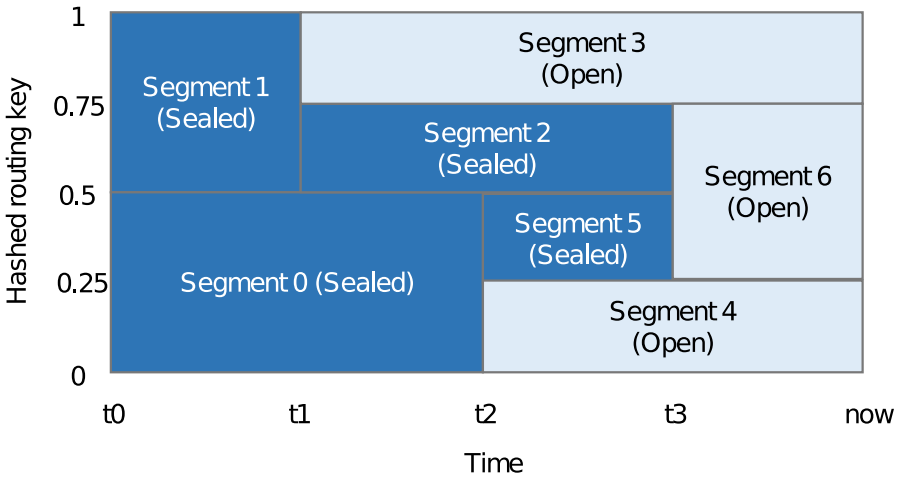


Figure 11: Example of stream auto-scaling process.

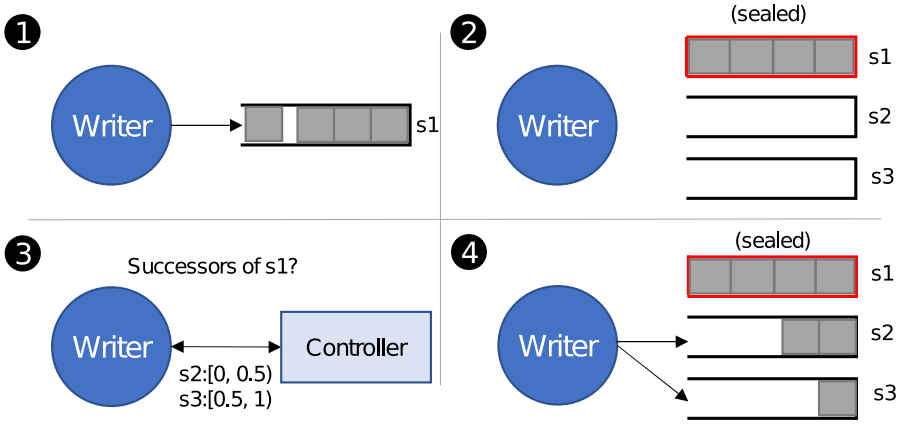


Figure 12: Controller-writer protocol upon scale-up.

tomatically change the segment parallelism of a stream based on the current load and a policy that determines when to scale up or down the number of parallel segments. While being a key feature, it also influences how writers and readers interact with the system. Therefore, it is important to understand how it works.

An example of stream auto-scaling is depicted in Fig. 11. A Stream starts at time $t0$ with two parallel segments. If the rate of data written to the stream segments is below the one defined in the scaling policy (e.g., 1MBps, 100 e/s), there will be no changes in the number of segments. However, at time $t1$, the data plane realized of a sustained increase in the ingestion rate for segment $s1$. When notified, the control plane *seals* $s1$ (no further writes are allowed) and *splits* it into two new segments (stream scale-up). Note that before $t1$, events written to the stream with a routing key k that hashes to $h(k) \in [0.5, 1)$ belong to $s1$, whereas events written to routing keys that hash to $h(k) \in [0, 0.5)$ belong to $s0$. After $t1$, events with routing key $h(k) \in [0.75, 1)$ are written to $s3$ and those $h(k) \in [0.5, 0.75)$ are written to $s2$. Fig. 11 illustrates another instance of a scale-up event on $s0$ at time $t2$. As one can infer, this process naturally distributes load across more segment containers.

Interestingly, segments covering a contiguous range of the key space can also be *merged* in the case they consistently receive low load (stream scale-down). For instance, at time $t3$, $s2$'s range and $s5$'s range are merged into a new segment $s6$ to accommodate a decrease in the load on the stream on that key range. As we show in the next section, Pravega allows clients to work with auto-scaling streams while preserving ordering and consistency guarantees.

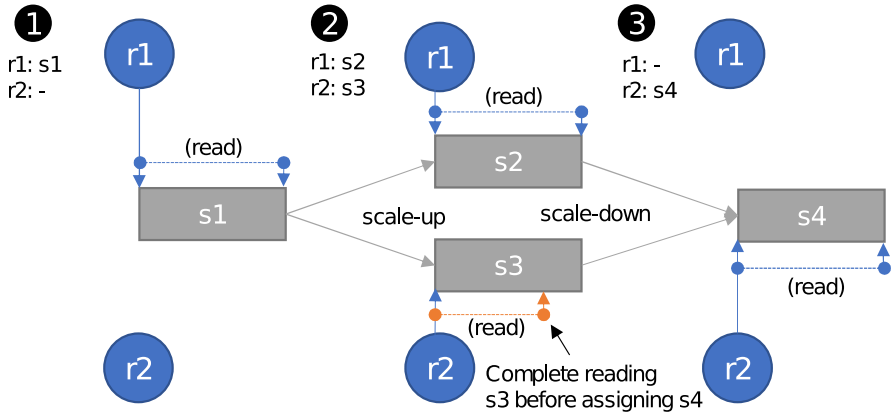


Figure 13: Reader group dynamics under auto-scaling.

Writing to a Stream. The Pravega writer interacts with the control plane to know the stream segments available to write at any given moment, as well as the segment store hosts that run the segment containers owning the segments. With this information, the writer can directly contact the right segment store host to write events to a given segment.

Pravega guarantees that events with the same routing key are read in the order they were written. To materialize this guarantee, Pravega enforces that the assignment of routing keys to segments is consistent even under stream auto-scaling. Between two stream scaling events, all events written to a stream with the same routing key are mapped to the same segment. The control plane builds the metadata that orders segments chronologically across scaling events. Let's retake the example in Fig. 11. The system scales up from one segment $s1$ to segments $s2$ and $s3$. The key space of $s1$ exactly overlaps with the ones of $s2$ and $s3$, but $s2$ and $s3$ have no intersection. In Pravega, segments $s2$ and $s3$ are defined as the *successors* of $s1$. As visible in Fig. 12, the protocol between the writer and the control plane enforces that no append happens to $s2$ and $s3$ until $s1$ is sealed, and this generalizes to any number of segments before and after a scaling event. Consequently, once segments are sealed due to a scaling event, future events are appended to the successors of the sealed segments, preserving routing key order.

Duplicates or missing events in a stream can be problematic: they can induce incorrect results or incorrect behavior in general. To avoid this problem, writers internally have a *writer id* used to determine the last event written upon a re-connection. When the writer has events to append, it initiates the write of a *batch of events*. Once finished appending the batch, the writer sends a "batch end" command with the number of events written and the last event number. The segment store must maintain the last event number for any given writer id to detect duplicates on segments. To this end, it persists the `⟨writer id, event number⟩` pair in a per-segment data structure called *segment attributes* as part of processing the append request. Upon a writer re-connection, the segment store fetches this attribute and returns the last event number written as part of the handshake with the writer. This response from the segment store enables the writer to resume from the correct event in the case it had appends outstanding.

Reading from a Stream. Reading a stream requires events to be processed only once, and consequently, a group of readers needs to coordinate the distribution of segments across the group. To enable multiple readers to read one or more streams in a coordinated manner, Pravega introduces the concept of *reader groups*. A reader group is a set RG of readers and an associated set of streams S, such that, for each $r \in RG, s(r) \subseteq \cup_{s \in S} c(s)$. At any time and for any two distinct readers $r, r' \in RG, s(r) \cap s(r')$ is empty. In this definition, $s(r)$ is the set of segments assigned to reader r, and $c(s)$ is the current set of active segments of a stream (non-sealed segments enabled for reading). Note that this definition does not imply that all segments in $\cup_{s \in S} c(s)$ are assigned to some reader at any time. It is possible that a reader has released a segment while no other has acquired it yet

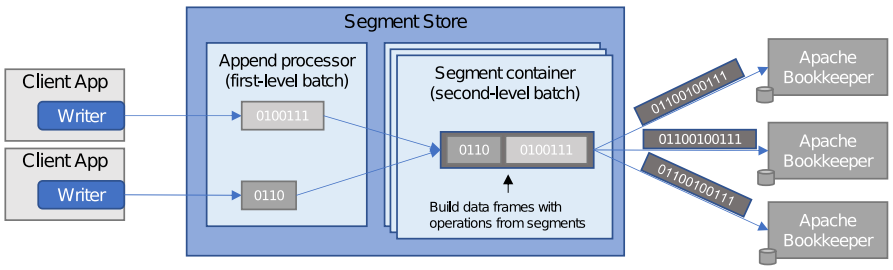


Figure 14: Overview of Pravega's write path and batching.

or a new segment has not been acquired yet by any reader. The contract specifies that any segment in $\cup_{s \in S} c(s)$ is eventually assigned. As such, the reader group does not guarantee that at any time $\cup_{s \in S} c(s) = \cup_{r \in RG} s(r)$, although we do guarantee for liveness that for all $x \in \cup_{s \in S} c(s)$, eventually x is assigned to some reader.

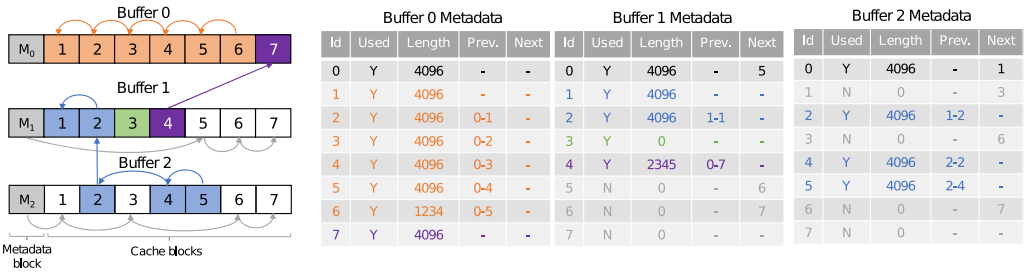
The assignment of segments to readers in the group is built upon the distributed coordination mechanism we expose in Pravega called *state synchronizer* [13]. The state synchronizer is an API built on top of Pravega streams that enables readers to have a consistent view of a distributed state via optimistic-concurrency. Readers use it to agree on changes to the state of the group; that is, the assignment of segments to readers. The distribution of segments attempts to achieve fairness (i.e., number of segments) across readers.

To guarantee that readers read events with the same routing key in append order, the readers follow a similar procedure as the writers. As an example, the stream in Fig. 13 has a single segment $s1$, and it eventually scales up, resulting in $s1$ splitting into $s2$ and $s3$. Once the reader $r1$ hits the end of $s1$, both $r1$ and $r2$ request the successors to the controller, and they start reading from the new segments $s2$ and $s3$. More interestingly, let's inspect what happens to readers $r1$ and $r2$ upon a stream scale-down event. At this point, $r1$ is reading $s2$ and $r2$ is reading $s3$. The segments merge into $s4$ ($s2$ and $s3$ are sealed). Reader $r1$ hits the end of $s2$ and request its successors, that is, segment $s4$. But reader $r2$ is not yet done with $s2$. If either $r1$ or $r2$ proceed to read $s4$ before $r2$ finishes reading $s2$, then we could be breaking our promise of reading events with the same key in append order. Consequently, to satisfy our order guarantee, we put $s3$ on hold until $r2$ flags that it is done with $s2$. Only then $s4$ can be assigned and read.

Pravega IO Path. We next describe the design decisions that allow Pravega achieving high IO performance irrespective of the event size and for high levels of parallelism.

Write Path. In Fig. 14, we illustrate the Pravega write path. Writers append applications' data and they batch such data to the extent possible. Conversely to other systems which batch data by holding it on the client and waiting to transmit it, the Pravega writer starts sending a batch before it has sufficient data to fill it and batch data is *collected on the server-side*. The writer controls batch sizes using a tracking heuristic that does estimates based on input rate and feedback from the responses. Specifically, the batch size is estimated as the minimum between the defined maximum batch size (e.g., 1MB) and half the server round trip time. With such estimates, the writer determines when to close batches. By doing this, the batch data is held not in client's memory, but is a mix of data in-flight on the wire and data collected at the server, thus reducing write latency.

In the segment store, every request that modifies a segment is converted into an *operation* and queued up for processing. There are multiple types of operations, each indicating a different modification to the segment (append, truncate, etc.). A segment container has a single, dedicated WAL log to which it writes all operations it receives. Many segments can be mapped to a single segment container, so all operations from a container's segments are *multiplexed into that single log*. This is a crucial design feature which enables Pravega to support a large number of segments, as it does not need to allocate physical resources on a per-segment basis.



Buffer 0 Metadata

Id	Used	Length	Prev.	Next
0	Y	4096	-	-
1	Y	4096	-	-
2	Y	4096	0-1	-
3	Y	4096	0-2	-
4	Y	4096	0-3	-
5	Y	4096	0-4	-
6	Y	1234	0-5	-
7	Y	4096	-	-

Buffer 1 Metadata

Id	Used	Length	Prev.	Next
0	Y	4096	-	5
1	Y	4096	-	-
2	Y	4096	1-1	-
3	Y	0	-	-
4	Y	2345	0-7	-
5	N	0	-	6
6	N	0	-	7
7	N	0	-	-

Buffer 2 Metadata

Id	Used	Length	Prev.	Next
0	Y	4096	-	1
1	N	0	-	3
2	Y	4096	1-2	-
3	N	0	-	6
4	Y	4096	2-2	-
5	Y	4096	2-4	-
6	N	0	-	7
7	N	0	-	-

Figure 15: A sample cache layout with 3 cache buffers (each with 8 4-KB blocks) and 4 cache entries stored (in colors).

The segment container aggregates multiple segment operations into a *data frame* and initiates the WAL write for it. When WAL acknowledges that write, the segment container asynchronously accepts all the operations in the data frame into its internal state. In this sense, the segment container builds a *second level of batching* by dynamically determining the size of data frames. Concretely, when the segment container sees that there are no more operations to pick from the processing queue, it uses recent WAL latency information and write sizes to calculate the amount of time to wait as follows: $Delay = RecentLatency \cdot (1 - \frac{AvgWriteSize}{MaxFrameSize})$. The delay to keep adding operations to a data frame is directly proportional to the recent WAL latency and inversely proportional to the recent average write size. If recent data frames had a high "fill rate", then the system is already maximizing throughput. On the other hand, if recent data frames were underutilized, it is desirable to wait a little longer (up to some defined bound), so that more operations may arrive and be batched together, potentially improving throughput.

WAL logs in Pravega are a metadata abstraction built on top of Apache Bookkeeper ledgers. A bookie, which is the Bookkeeper storage server, journals requests to append data to a ledger, and it performs another level of aggregation before appending to its journal. This third level of aggregation is another opportunity to batch data coming from different segment containers.

Read Path. Readers issue read requests to segment store instances. In this sense, the *read index* is an essential component of the segment container that provides a transparent view of all the data in a segment, both from WAL and LTS, without the external caller having to know where such data resides. The read index can access segment data in a random fashion and contains an entry per active segment in the segment container. When a read request is received, the read index returns a read iterator that will return data until the read request parameters are satisfied. The iterator will either fetch data that is immediately available in the in-memory cache, or request data from LTS (and bring it to cache) or, if it reached the current end of the segment, return a future that will be completed when new data is added (thus providing *tail reads*). At the heart of the read index lies a sorted index of entries per segment (indexed by their start offsets) which is used to locate the requested data. The index is implemented via a custom AVL search tree with a goal of minimizing memory usage while not sacrificing insert or access performance. The entries themselves contain some small amount of metadata that is used to locate the data in the cache and to determine usage patterns to favor cache evictions.

The read index is backed by a local *in-memory cache* designed from scratch for Pravega. In our experience, traditional cache solutions treat each entry as an immutable blob of data, which poses problems for the append-heavy ingestion workloads that are common in streaming scenarios. Each event appended to a stream would either require its own cache entry or need an expensive read-modify-write operation to be included in the cache.

We divide our cache into equal-sized cache blocks, where each block is uniquely addressable using a 32-bit pointer. Cache blocks are daisy-chained together to form cache entries. Each cache block has a pointer to the block immediately before it in the chain. Since each block has an address, we can choose the address of the last block in the chain to be the address of the entry itself. We can

then reference this address from the read index. While a bit counter-intuitive, pointing to the last block enables us to immediately locate that and perform appends, by either writing directly to it (if it still has capacity) or find a new empty block and add that to the chain. Similarly to the blocks used in cache entries, empty cache blocks are also chained together, which makes locating an available block an $\mathcal{O}(1)$ operation. To prevent memory fragmentation and keep metadata overhead low, the cache pre-allocates during initialization contiguous regions of off-heap memory into cache buffers (e.g., a 2MB buffer can hold 512 4KB blocks). Regarding empty cache blocks, keeping a single list of such blocks across all buffers would would quickly run into concurrency issues while modifying it. Thus, we have chosen to only keep a list of empty cache blocks within each buffer (smaller concurrency domain). Across buffers, the cache uses a queue of cache buffers with available blocks that are added and removed based on whether they have available blocks or not. For clarity, Figure 15 depicts a cache with four entries (colored blocks) and three buffers following the described cache layout, along with a tabular representation of the metadata.

Storage Tiering. WAL is by no means the final destination of data. Instead, one of the key goals of Pravega is to *asynchronously move data to LTS* (e.g., S3, HDFS, NFS) for cost and throughput reasons. And it does so in a unique manner: to wit, the storage tiering process is *integrated in the write path*. This implies that if LTS is not available or is temporarily too slow, Pravega is able to throttle writers to prevent backlogs of data to grow indefinitely waiting to be moved to LTS.

In the segment container, the *storage writer* is the component in charge of de-multiplexing the operations written to WAL, grouping them by segment, and applying them in LTS. It performs several optimizations along the way to maximize throughput, such as buffering smaller appends into larger writes. Once the storage writer flushes a set of operations to LTS, it notifies the segment container that the WAL log can be truncated up to that point. This translates into deleting Bookkeeper ledgers when needed.

The storage writer interacts with the storage subsystem in charge of writing data to LTS and keep the metadata of segments in LTS. In LTS, Pravega stores *chunks* (i.e., contiguous range of segment bytes) and segments are made up of a sequence of non-overlapping chunks. Note that chunks themselves do not include additional metadata. Similar to the case of the control plane storing the metadata of streams, the metadata of chunks in LTS is also stored in Pravega itself via key/value tables API. All LTS metadata operations are performed using conditional updates and using a single transaction that updates multiple keys at once. This guarantees that concurrent operations will never leave the metadata in inconsistent state.

Failure Handling. Pravega should assume the occurrence of failures in its own instances or in its dependencies. First, segment containers adopt a *fail fast* approach: if a severe error is detected within a segment container (e.g., out of memory) or with a dependency (e.g., Bookkeeper or LTS are unreachable), the segment container shuts down. This implies that no further operation is allowed and it attempts to perform a *recovery*. A recovery is part of the initialization process of a segment container and it consist of i) starting its internal components, and ii) read the WAL log to rebuild the internal state just before the crash. In fact, the main goal of WAL is to persistently store the operations not yet written to LTS to recover the segment container's state. It is worth mentioning that the segment container periodically writes a special operation to the WAL log called *metadata checkpoints*, which are a snapshot of the segment container metadata at a given point in time. To recover its state, the segment container just needs to read the last metadata checkpoint and sequentially apply the subsequent operations in the WAL log.

If a whole segment store instance crashes, all the segment containers it was running are redistributed across the remaining instances. In this scenario, there can be cases in which more than one segment store instance attempts to run the same segment container, thus leading to potential data corruption. Pravega guarantees that a segment container writes data to WAL and LTS exclusively at any moment by: i) Pravega keeps the assignment of segment containers to segment stores in a consistent store (i.e., Apache Zookeeper); ii) Segment containers implement a technique called "fencing" to ensure exclusive access to WAL logs[3].

[3] `https://bookkeeper.apache.org/docs/development/protocol`

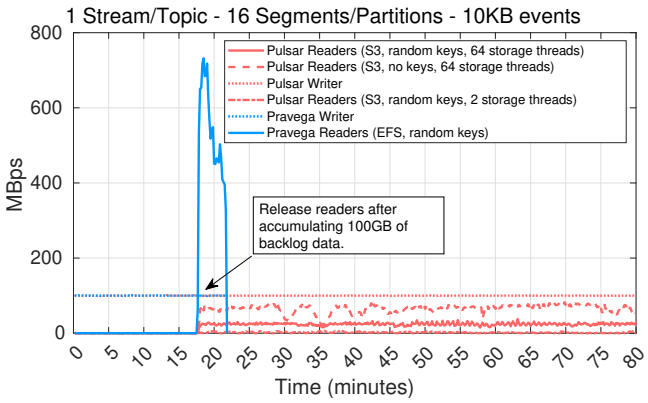


Figure 16: Historical read performance of Pravega compared with Pulsar. In this experiment executed on AWS, writers write 10KB events until generating a 100GB data backlog. At this point, readers are released and they are expected to catch up with writers by reading a large fraction of data from a external storage system. Visibly, the approach in which Pravega implements storage tiering provides much higher historical read performance than Pulsar.

Streaming Connectors. Pravega offers a wide range of connectors to manage stream data from data processing engines. The most relevant connectors for the project are the following ones:

- *GStreamer Connector*[4]: GStreamer is a pipeline-based multimedia framework that links together a wide variety of media processing systems to complete complex workflows. Pravega provides a GStreamer sink and source to allow GStreamer multimedia flows to be stored and served via Pravega. This connetor will be extensively used in the computer-assisted surgery use-case with NCT Dresden.
- *Flink Connector*[5]: The Flink Connector enables building end-to-end stream processing pipelines with Pravega in Apache Flink. This also allows reading and writing data to external data sources and sinks via Flink Connector. We will work with this connector to perform stream processing tasks with Pathogen workloads provided by UKHSA.
- *Spark Connector*[6]: Connector to read and write Pravega Streams with Apache Spark, a high-performance analytics engine for batch and streaming data. The connector can be used to build end-to-end stream processing pipelines that use Pravega as the stream storage and message bus, and Apache Spark for computation over the streams.

Among other tasks, in the context of the project, we will work on improving, extending, and exploiting these connectors to satisfy the needs of NEARDATA use-cases.

Integration of Pravega with NEARDATA and use-cases. Let us revisit the role that Pravega plays within the NEARDATA use cases and how it will be of great help to our use-cases:

- *Integration with object storage*: Pravega is the only streaming storage system that fully integrates the movement of stream data to an external storage in the write path. This means that Pravega was designed since day 0 for migrating small events coming into the system to larger chunks to be eventually stored in external storage, like object stores. The fact that NEARDATA is built on top of object storage provides a natural alignment between the project and the original design of Pravega. To give a sense on this, we refer to Fig. 16. As can be observed, Pravega provides a

[4] `https://github.com/pravega/gstreamer-pravega`
[5] `https://github.com/pravega/flink-connectors`
[6] `https://github.com/pravega/spark-connectors`

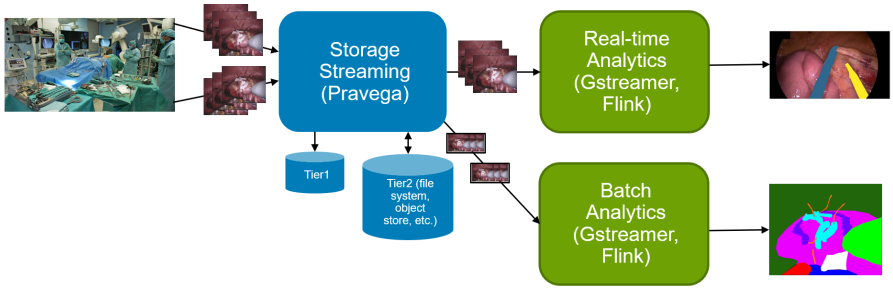


Figure 17: Integration of Pravega with NCT use-case.

better historical read performance than Pulsar due to their different implementations of storage tiering. NEARDATA use cases will benefit from high historical read performance by exploiting high throughput and parallelism of object stores when executing batch analytics on historical stream data via Pravega.

- *Integration with use-cases*: Pravega will allow our use-cases requiring stream analytics to transparently exploit object storage. As depicted in Fig. 17, NCT requires low latency video analytics on real-time video being collected in operating rooms during surgery. Such video is being generated in real time and needs to be durably stored and processed. Pravega satisfies the storage requirements, while the GStreamer connector for Pravega allows us to read and write video streams from Pravega, as well as feeding video analytics engines with such data. But, at the same time, NCT may also need to read old surgery video for training analytics models. Pravega unifies access to recent and historical stream data under the same APIs, so NCT data scientists will be able to access historical stream data with good performance. Another interesting use-case that will benefit from Pravega is UKHS. UKHS generates large amounts of genomic sequencing files from pathogens. Naturally, the rate of genomic files being generated may vary along time depending on daily patterns or sudden virus outbreaks. In this regard, Pravega streams have a unique property: they are elastic. That is, the parallelism of a Pravega stream (i.e., number of parallel segments or partitions) can change based on the load received and the scaling policy configured by the user. This yields that Pravega streams can adapt their parallelism to the workload fluctuations of pathogen files generated by UKHS. We believe that Pravega can be an excellent fit for the UKHS workload as it may i) exhibit significant fluctuations, ii) requires fast results, and iii) uses a file format that can be exploited by streaming computations (e.g., FASTQ).

- *Integration with serverless frameworks*: Pravega provides a wide variety of useful APIs in addition to the streaming ones (e.g., state synchronizer, K/V, atomic variables). In fact, DELL and URV are exploring the suitability of Pravega APIs for serving stateful serverless analytics. This collaboration will help to further integrate the streaming and serverless fabrics for NEARDATA, which will certainly provide a solid integration of the core data plane components of the project.

Research challenges ahead. During the project, we will investigate several aspects related to our use-cases that go beyond the state-of-the-art in streaming storage systems. i) *Coordination of data stream and processing parallelism*: Pravega streams are elastic and can auto-scale based on policies if the ingestion workload fluctuates, as may happen with the rate at which genomic data files are generated in UKSHA. However, if we think of a stream processing pipeline, the parallelism of the processing engine may need to change accordingly. Unfortunately, finding practical and effective ways of coordinating auto-scaling of source data streams and processing engines is still an open question. ii) *Data streams as storage substrate for FaaS pipelines*: As of today, FaaS pipelines normally exchange results using object storage. Recently, some research works started to explore using write-ahead logs for FaaS pipelines [14]. We believe that streaming storage systems can be the next natural storage substrate

for FaaS pipelines and Pravega may have a key role in this regard. We will need to understand the trade-offs of using data streams for FaaS pipelines, as well as novel Pravega APIs that can improve programmability in this scenario. iii) *Multi-dimensional stream content indexing*: Streaming storage systems or messaging systems can normally use "time" as a dimension to index a data stream. While this is useful, we believe that we could index streams on arbitrary dimensions related to its contents. For instance, it would be useful for NCT to index the points in a video stream where a tumor is identified, so researchers can quickly locate that part of the video and work with it. This implies setting up mechanism that "identifies" the object to index, and interacts with the streaming storage system to efficiently build that index. We will investigate practical and extensible mechanisms to implement this, as it would be of great help for our use-cases, specially regarding video analytics. iv) *Data reduction in streaming storage systems*: Currently, systems like Apache Kafka and Apache Pulsar just provide simple and static client-side data compression for reducing storage consumption. However, we believe that there could be multiple other options to perform data reduction in streaming storage systems like Pravega, as this topic remains largely unexplored.

2.5 Control Plane components

The Data Broker provides a user authentication service that grants access to confidential data, which can be decrypted and manipulated. Additionally, the Data Broker includes orchestration services and connects different types of data streams through analytical platforms and services. First, we discuss the Confidential Compute layer with confidential technologies, then the Confidential Data Exchange layer, in which we focus on how the data broker accesses and discovers confidential data, and finally, in the Confidential & Federated orchestration section, we elaborate the declarative interconnection framework for extreme data workflows that ensures confidentiality and security throughout the entire data path.

2.5.1 Confidential Technologies

SCONE framework. When it comes to securely exchanging and managing confidential data, we propose utilizing SCONE, a framework that simplifies the utilization of Trusted Execution Environments (TEEs) [15] like Intel SGX [16]. The objective here is to protect data at rest, i.e., stored on persistent storage, data in use, i.e., while being processed, and data in transit, while it is exchanged/transferred between nodes. Technologies such as Intel SGX enable users to isolate their processes such that even privileged users cannot access the encrypted memory region ensuring confidentiality and integrity for the data being processed.

Different levels of abstractions for such TEEs exist. While Intel SGX provides isolation on function-/process-level as shown in Figure 18, other approaches such AMD SEV provide such protection on VM-level. Each of these abstractions have different advantages and disadvantages such as the size of the trusted computing base vs. configure-ability, and ease of orchestration. Furthermore, the use of such technologies requires either binary modifications or re-compilations making their use challenging for novice users.

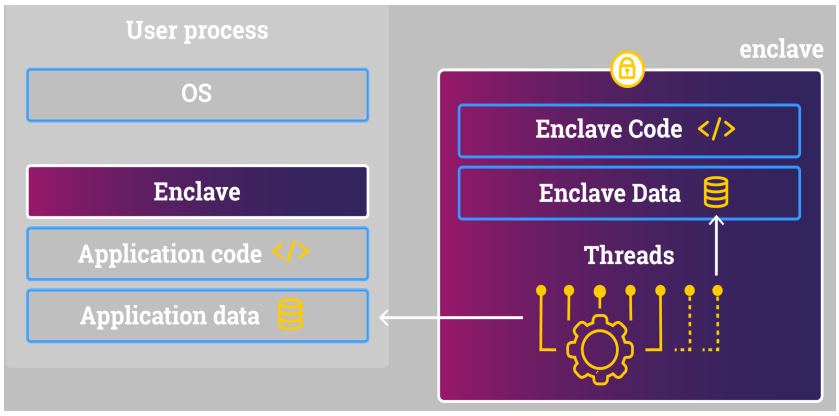


Figure 18: Isolation on process-level using Intel SGX

In order to use TEEs, developers have to be furthermore familiar with the different hardware capabilities in order to utilize the provided features by enriching the existing source code such that functions or the whole executable will run in trusted compartments, i.e., enclaves.

Within NEARDATA, we propose and envision a transparent approach where native applications are transformed into confidential ones using the **SCONE tool chain**. This approach will be furthermore complemented by an **automatic configuration** which ensures that processes exchange data in a secure fashion including mutual authentication. Furthermore, we propose **transparent file system encryption** that neither requires any modification of the existing code base nor the use of overlay file system encryption techniques such as provided by fuse [17] etc. This approach will ensure that encryption and decryption will be carried out within the trusted compartment providing the desired confidentiality and integrity aspects.

Network and File System Protection. In addition to the components described above, we envision the use of shields provided in SCONE. In Figure 19 gives an overview of the shields that target the prevention of low-level attacks and ensure the confidentiality and integrity of shared data in NEARDATA. We plan to utilize the following two shielding mechanisms that SCONE provides.

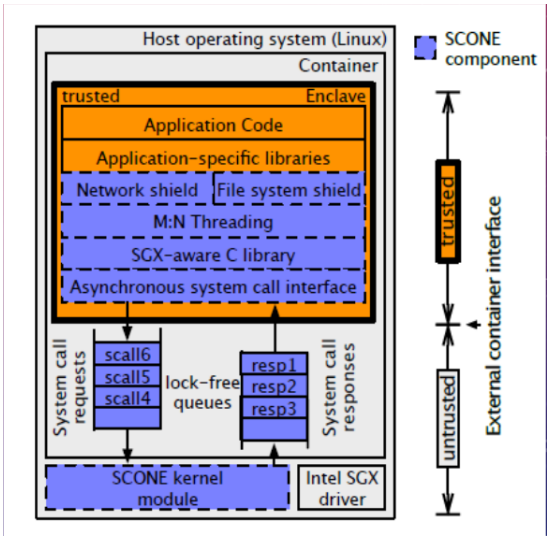


Figure 19: SCONE Architecture-Interface Shielding

The main objective of the shields is to:

1. Prevent low-level attacks, such as the OS kernel controlling pointers and buffer sizes passed to the service

2. Ensure the confidentiality and integrity of the application data passed through the OS.

The shields to be used are as follows: **File System Shield** ensures the confidentiality and integrity of files. In the file system shield, a container image creator must define three disjoint sets of file path prefixes: prefixes of (1) unprotected files, (2) encrypted and authenticated files, and (3) authenticated files. When a file is opened, the shield determines the longest matching prefix for the file name. Depending on the match, the file is authenticated, encrypted, or just passed through to the host OS. The file system shield splits files into blocks of fixed sizes. For each block, the shield keeps an authentication tag and a nonce in a metadata file. The metadata file is also authenticated to detect modifications. The keys used to encrypt and authenticate files as well as the three prefix sets are part of the configuration parameters passed to the file system shield during startup. **Network Shield**. In order to protect the confidentiality of secrets transferred from CAS to computation components running inside enclaves, we need to protect the network communication between CAS and the components to make sure that an attacker cannot observe the network traffic to steal the secrets. To achieve this goal we plan to utilize the network shield for data processing components that wraps the communication between all peers and ensures that all data passes to the connection is TLS-encrypted. The certificates for TLS connections are saved in a configuration file protected by the file system shield mentioned previously.

SCONE Attestation. Service (CAS) technologies. Three components have been introduced here: **CAS** – configuration and attestation service, – which provides the means to tell whether an application is trustworthy or not; **Keycloak** – an identity and access manager, – that generates access tokens with configuration to allow or deny access to resources and validates them; and the **Certifier** – certificate and private and public keys generator, – that can be used by applications that do not have a TEE available.

CAS is responsible for registering the policies (a.k.a. sessions) and their subsequent validation, such that an application can run attested. It is a key decision-maker, that tells whether an application can be trusted or not. Policies comprise a set of configurations that only an attested application can read. Elements such as the specific command-line parameters to load the system, environment variables that are not subject to external influence, private keys and certificates, and configuration files. For example, if the same configuration file is within the policies and also present in the filesystem, the latter is ignored by the attested application and the former is used invariably.

The CAS performs the following tasks:

- Code attestation - local or remote.
- Configuration provisioning.
- Management of SCONE policies.
- Perform configuration of the target application.
- Provision secrets.
- Configure network as well as file system encryption/shielding.

The Figure 20 image describes the high-level architecture of the SCONE Runtime as well as the interaction of CAS. Note that the SCONE runtime is weaved into the binary code of the microserver/application.

In the following section, we will provide additional information regarding the functionality and security guarantees of SCONE CAS. Following a brief overview of CAS's purpose, we describe its various functionalities, such as key generation and management and access control. SCONE CAS

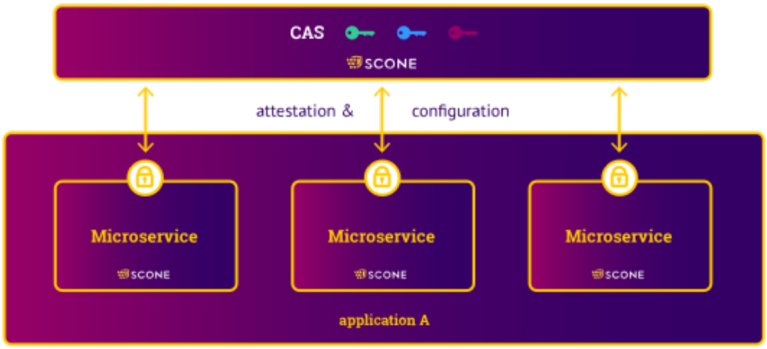


Figure 20: SCONE CAS (Configuration & Attestation Service)

manages an application's secrets, particularly its keys. Only services with explicit permission from the application's policy are granted access to keys, encrypted data, encrypted code, and application policies. SCONE CAS can generate keys on an application's behalf. The generation occurs within a secure execution environment. Access to keys is governed by an application-managed security policy. The keys and security policies are inaccessible to root users and SCONE CAS administrators. Currently, SCONE CAS operates within SGX enclaves.

Access control mechanism use to modify or read a policy, a client must demonstrate, via TLS, that it possesses the private key corresponding to a policy-specified public key. SCONE CAS grants these clients access to this policy without exception. Typically, the client's access to a private key is also governed by a policy, possibly the same policy. Note that a client can only access their private keys following a successful attestation.

In Management, SCONE CAS administration can be delegated to a third party. CAS itself ensures the confidentiality and integrity of the policies and their secrets. Since the entity that creates a policy has full control over who can read or modify it, no administrator managing the CAS can alter the application's access control to a policy.

Encrypted Code, On a trusted host, one can generate images using encrypted Python, Java, JavaScript, C#, or any other JIT or interpreted code. This code could alternatively be generated within an enclave. One can verify and decrypt the code contained within an enclave transparently. This is possible without changing the Python engine or the Java, etc. virtual machine. Notably, SCONE CAS certifies both the Python engine and Python code.

Keycloak is an open-source approach for identity and access management that focuses primarily on applications and services. Users can authenticate via Keycloak as opposed to specific applications. Therefore, applications are not required to deal with login forms, authenticating users, or storing users. Once a user has authenticated into Keycloak, they are not required to sign in again to access other applications. The Keycloak is an additional mechanism for enforcing the authentic identification of system operators within NEARDATA and their corresponding access level to other systems and resources. Keycloak will issue access tokens with the particular responsibilities that a system administrator has assigned to the requesting user. The other system that receives this token will obtain a validation token from Keycloak that attests to its authenticity and verifies that the roles contained within are accurate.

The Keycloak is safeguarded by confidential computing mechanisms, such as an enclave combined with SGX, which ensures confidentiality. It also has an added layer of protection through CAS, which ensures that all configuration, credentials, and keys are securely provisioned via attestation. It is additionally protected by CAS, so all configuration, credentials, and keys are provisioned via attestation. Access tokens and validation tokens are issued via REST communication using an HTTPS server with a TLS-enabled secure channel.

Certifier is a RESTful application that returns a payload consisting of a certificate, its private key,

and its corresponding public key. If an application is running on hardware without a TEE to authenticate and obtain a certificate and private key from CAS-registered policies, you can use Certifier in a "para-attestation" setup. This allows adjacent applications to be trusted, even if they don't meet the same level of trustworthiness required for handling sensitive data.

2.5.2 Confidential data exchange

Threat Model. In NEARDATA we target to harness Edge cloud resources as well as edge devices to conduct data analytic tasks in addition to traditional cloud resources, which shifts a significant part of the workload from the cloud to edge devices. Due to the rapid growth in streaming data volume, many applications are leveraging edge cloud platforms these days to efficiently store and analyze data. Such platforms must be trusted to protect data privacy and security from malicious insiders or outside attacks when hosting applications in these environments.

For the threat model, we consider classical cloud application as well as edge platform services capturing and analyzing, e.g. telemetry data. We also recognize the significance of mission-critical IoT with tight control loops but do not target them in our threat model. Our target scenario includes source sensors, edge platforms, and cloud servers. We assume a potentially malicious environment in which privileged processes such as the operating system have full control over system call arguments and their results. In such a compromised system, an attacker can not only modify the system data but can also eavesdrop on system activities. Apart from that, we assume that access to the hardware is strictly regulated and that an adversary cannot mount physical attacks on the otherwise trusted CPU.

In addition, we consider the scenarios of malicious field nodes trying to access other users' resources and malicious remote nodes trying to access data from field IoT devices in addition to regular cloud nodes. The first scenario includes legitimate, but compromised, nodes trying to access resources and applications in the edge node or in the cloud for which they are not authorized. In the second scenario, an external attacker tries to access data generated by devices in the field or even take control of the devices. Also in this scenario, the attacker can be authorized to access some data but wants to access data or perform actions for which they have no authorization. We furthermore assume untrusted edge-cloud links that require encryption of uploaded data. We consider malicious adversaries capable of identifying IoT data, tampering with edge processing outcomes, or obstructing processing progress as in-scope threats. Based on the assumption that powerful adversaries exist: they control all applications on the edge by exploiting weak configurations or bugs in the edge software; they control the entire OS as well as all applications on the edge. For cloud nodes, we assume an adversary has full control over all processes and all hardware components except the CPU, based on the Intel's SGX threat assumption. The adversary can check the Memory trace of all processes except those running within the enclaves. Also, they can monitor or interfere with the communication between edge servers and cloud servers.

Attacks In The Context of NEARDATA. Considering the above-listed threats, a malicious user can drive the following attacks in the context of NEARDATA applications: In order to gain access to sensitive data that is used for training purposes such as radiomics imaging data, etc., an infrastructure administrator with root privileges can simply copy the locally stored files out of the running VM image. Although this attack can be prevented by using end-to-end encryption, i.e., encrypting the files beforehand, the Python processes running the training software such as Pytorch, TensorFlow, etc., need to access this data, i.e., require access to the private key used to encrypt the data at rest. An attacker can therefore create a memory dump of these processes in order to reveal the key and perform the en/decryption him-/herself.

It is also worth noting that training data is a precious resource that can be monetized and conveyed to multiple entities also in real time directly from the edge. As a variation of the above attack, an external malicious user can attempt to access data from the field devices by exploiting their low complexity and lack of support for fine-grained access control policies. In NEARDATA, we tackle this attack by means of a Data Provider node, which intermediates any exchange between the de-

vices and the users.

Besides gaining access to confidential data, another attack vector is the malicious introduction of wrong information in order to tamper with training results. This can be achieved by a malicious user pretending to be a legitimate collaborator if we consider federated learning. Although this attack seems to be not that easy at first as it requires access to certificates as well as keys in order to pass the mutual authentication when establishing TCP connections between the collaborator as well as the aggregator, such keys, as well as certificates, can be easily retrieved as described previously.

Another way of tampering with training results is through the modification of the training code itself. This type of attack can be prevented through the use of integrity protection mechanisms at the file system level such that the code is signed beforehand. Another type of attack are so-called rollback attacks: For these attacks, a malicious user provides the software with an older version of either the training data or the trained model such that, e.g., classifiers do not correctly detect/recognize certain items any more. This requires, as before, access to the file system as well as the capability to stop processes and resume them which is easily achievable by administrators with root privileges and hypervisor access.

In stream processing systems, network communication patterns directly reflect the structure of the streaming applications. These applications typically consist of multiple processing stages organized into a Directed Acyclic Graph (DAG) that runs on a collection of networked machines. In general, each stage is partitioned into multiple nodes that are executed in parallel. Each node performs local computations on the input streams from its in-edges and produces output streams to its out-edges. By observing network-level communication between the different stages of the DAG, an adversary may be able to extract information about the data being processed by the application. In addition, stream processing systems are susceptible to side-channel attacks, which compromise users' data security and privacy using any publicly accessible information that is not privacy-sensitive in nature, namely side-channel information. Such public information is typically correlated "secretly" with certain privacy-sensitive data that should be protected. Attackers then explore the hidden correlations to finally infer the protected data from the side channels. Since any public information can have the potential to link to some sensitive data, side-channel attacks can happen anywhere in the edge computing architecture. These side-channel attacks happen both at the memory level and network level.

Confidential Data Exchange through Data Broker. This section describes how sensitive and encrypted data is accessed and decrypted by the data broker. Based on the NEARDATA architecture as depicted in [fig 2 of the Proposal, a UML components diagram [fig. 21]is presented to guide the development, configuration, modification, and deployment of artifacts in relation to their interaction with confidential computing resources.

In this context, CAS, Keycloak, and Certifier are utilized as mentioned earlier in the section. As we mentioned in the previous section the Keycloak is an Identity and Access Management system (IAM) that provides means to control identities and their corresponding permissions within specific domains of serviced applications. The sequence diagram [fig 22] represents an entity (a person or an application on behalf of this person) using an application and requesting its roles from the IAM The application, in turn, will check the roles provided by this entity and validate them against the IAM. The serviced application will check with the IAM the information the entity has provided and allow or deny the requested access, accordingly. This workflow will be aborted and the entity will have denied the service if: the entity cannot identify itself to the IAM; or if it doesn't have appropriate roles; or if, having the roles, the IAM does not recognize its validity.

As mentioned in the previous section the Certifier can be employed in an arrangement that adjacent applications can be trusted upon a method called "para-attestation", whence the trustworthiness of these adjacent applications is lenient to the extent of what sensitive information they are allowed to handle. [fig 23] shows an example of how a Data Connector can employ such a mechanism.

The three new elements introduced above not only enable confidential computing in a distributing systems environment (with CAS), but also offer alternatives to enforce accountability and trace-

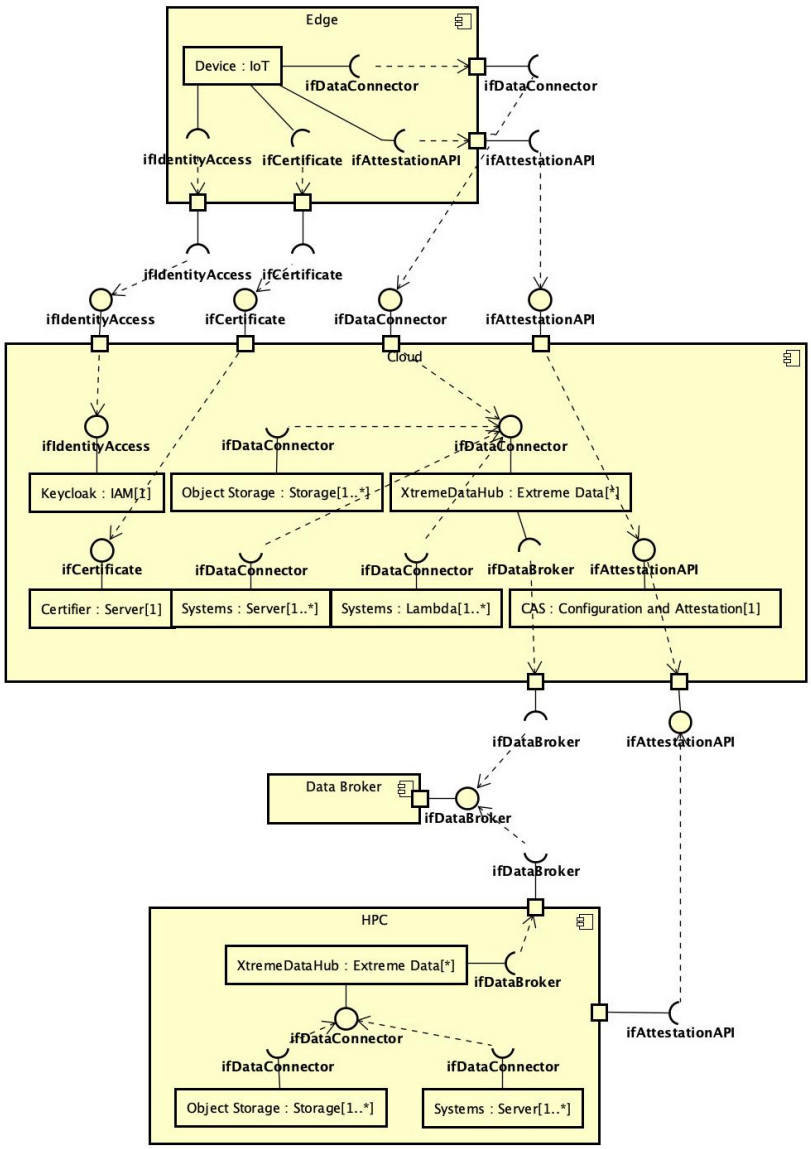


Figure 21: Components diagram representing confidential computing resources

ability (with Keycloak) – that can again be improved with auxiliary applications. It also offers the means to enable confidentiality in applications running in the *edge* side, near the data being originated, *i.e.* in hardware that doesn't support TEE (especially SGX). They are all running on the cloud side [fig. 21] as confidential computing applications, therefore attested executions of trustworthy systems. And this is essential to the method of *para-attestation* to work and to be trusted in conjunction to the *Policy Boards* to determine the extent of trustworthiness of edge endpoints.

2.5.3 Confidential orchestration

Data Broker provides confidential orchestration services and connects different data streams through analytical platforms and services. With regards to orchestration, we envision leveraging a container-based approach based on state-of-the-art **container-runtimes** such as Docker [18]. This will allow for easy orchestration of the different service entities through modern **orchestrators** such as Kubernetes. Furthermore, we will investigate the use of service meshes such as Istio [19] for confidential data exchange handled transparently to the orchestrated applications. This will impose several challenges such as automatic configuration as well as seamless integration such that no modifications to the original application and services are required while still providing the highest level of protection.

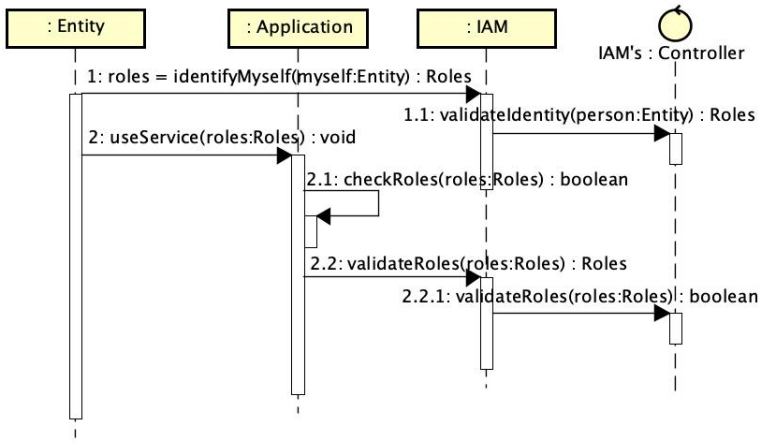


Figure 22: Sequence diagram representing a simplified workflow of authorization of services usage upon proof of identity

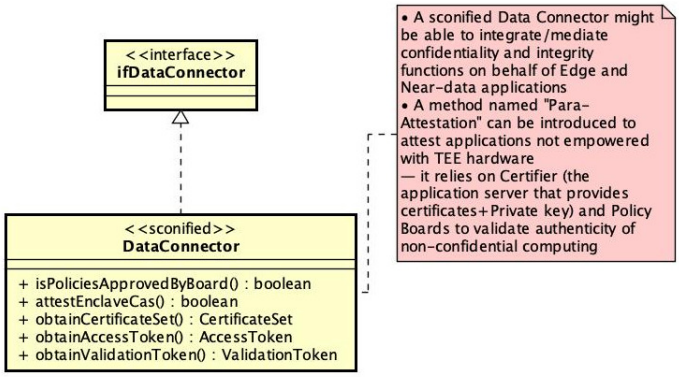


Figure 23: Data Connector class implementing methods of the Data connector interface

Confidential execution of Apache Spark pipelines. The rapid growth in streaming data volume has prompted many applications to utilize edge cloud platforms for efficient storage and analysis of large amounts of data. While deploying applications in the edge cloud requires trusting cloud platforms to safeguard data security and privacy against malicious insiders and external attackers. In order to process data securely in the cloud, hardware-based approaches, such as Intel SGX, are the most practical solution, but we need to address several challenges when processing streaming data applications on Intel SGX to provide databroker-like capabilities. The streaming data analytics are extremely memory-intensive, since these systems are almost always based on Java Virtual Machine (JVM). Intel SGX still suffers from side-channel attacks. Moreover, Intel SGX requires users to heavily modify the source code of their application to run inside enclaves. Therefore, transparently supporting an unmodified distributed streaming data analytics framework to run inside enclaves is not trivial.

To overcome these challenges we envision integrating PySPARK with SCONE. It consists of two main components: (i) Configuration and attestation service (CAS) component that will carry out authentication, authorization and secret injection, and (ii) PySpark with integration with the SCONE runtime to run inside enclaves. The objective is to execute only sensitive parts, i.e., the computation parts that process the input-sensitive data, inside enclaves as well as to control data sharing. The computation parts outside of enclaves can only access encrypted data. Hence, the driver and the Python processes execute inside Intel SGX enclaves using SCONE. Moreover, when utilizing PyS-

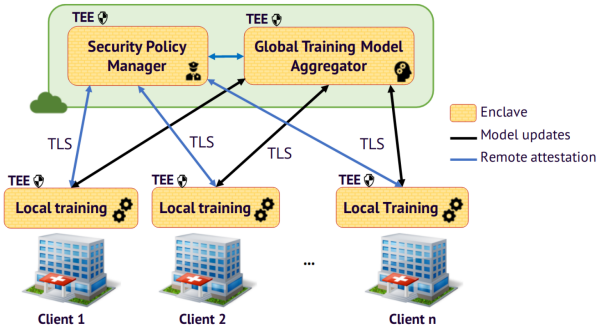


Figure 24: Federated Learning Architecture

PARK, we need to ensure the confidentiality and integrity of the driver as it is responsible for splitting and scheduling tasks in the system. This can be achieved by protecting the Python processes since they are the sensitive computation parts which directly decrypt and process the sensitive input data. Note that within this approach, we would not only encrypt the input sensitive data but also the computation over it (Python code of analytics jobs). In addition, we envision combining this with remote attestation to ensure the code and data integrity.

When utilizing remote attestation, we need to make sure that the shared secrets such as certificates, passwords to launch data processing substrates such as PySpark, or keys for encrypt/decrypt the input data and computations will never be revealed to untrusted components. To achieve these goals, we envision the use of a configuration and attestation service (CAS) that securely transfers secrets only to the components that have authenticated themselves successfully against it. The CAS enhances the Intel attestation service to bootstrap and establish trust across the machines running this platform and maintain a secure configuration for the system. In detail, CAS remotely attests the driver and worker processes of such a process running inside enclaves, before providing encryption/decryption keys and other configuration parameters. The CAS itself runs inside a TEE such as an Intel SGX enclave. A user of this platform first encrypts the input data and his data analytics job and then uploads these secrets (cryptographic keys) and system configurations to the CAS. Using policies, it is furthermore possible to control access We also envision designing and implementing an auditing service in CAS to keep result logs during runtime for accountability. This auditing service targets the protection of the data analytics system against rollback attacks.

Confidential Orchestration of the federated environment. The orchestration service may leverage Kubernetes orchestration mechanisms but also higher level workflow definitions such as systems like FedML (Federated Learning). Federated Learning strategies that employ SCONE in the context of Machine Learning using data from multiple healthcare centers. This is primarily associated with NCT. The architecture of confidential federated learning can be observed in Figure 24. The main goal of the layer is not only to ensure the confidentiality, integrity and freshness of input data, code, and machine learning models but also to enable multiple clients (who do not necessarily trust each other) to get the benefits of collaborative training without revealing their local training data.

The main goal of our hardened version is not only to ensure the confidentiality, integrity and freshness of input data, code, and machine learning models but also to enable multiple clients (who do not necessarily trust each other) to get the benefits of collaborative training without revealing their local training data. In the confidential setup, each client performs the local training also inside TEE enclaves to make sure that no one tampers with the input data or training code during the computations. To govern and coordinate the collaborative machine learning training computation among clients, we design a trusted management component, called Configuration and Attestation Service which maintains security policies based on the agreement among all clients to define the access control over global training computation, the global training model, also the code and input data

used for local training at each client. The Configuration and Attestation Service automatically and transparently perform remote attestation to make sure the local computations are running the correct code, correct input data, and on the correct platforms as per the agreement. It only allows clients to participate in the global training after successfully performing the remote attestation process. It also conducts the remote attestation on the enclaves that execute the global training in a cloud, to ensure that no one at the cloud provider side modifies the global training aggregation computation. In addition to remote attestation, it encrypts the training code, and the Configuration and Attestation Service only provides the key to decrypt it inside enclaves after the remote attestation. Secrets including keys for encryption/decryption in each policy are generated by the Configuration and Attestation Service also running inside Intel SGX enclaves and cannot be seen by any human or client. Examples of the policies can be found in [20, 21].

After receiving the agreed security policies from clients, the Configuration and Attestation Service strictly enforce them. It only passes secrets and configuration to applications (i.e., training computations), after attesting them. The training computations are executed inside Intel SGX enclaves and associated with policies provided and pre-agreed by clients. The training computations are identified by a secure hash and the content of the files (input data) they can access. Secrets can be passed to applications as command-line arguments, or environment variables, or can be injected into files. The files can contain environment variables referring to the names of secrets defined in the security policy. The variables are transparently replaced by the value of the secret when an application that is permitted to access the secrets reads the file.

We design the Configuration and Attestation Service in a way that we can delegate its management of it to an untrusted party, e.g., a cloud provider, while clients can still trust that their security policies for protecting their properties are safely maintained and well protected. In the confidential version of the federated learning application, clients can attest to the Configuration and Attestation Service component, i.e., they can verify that it runs the expected unmodified code, in the correct platform before uploading security policies.

We implemented the federated learning prototype using Intel OpenFL [22]- a distributed federated machine learning framework. We ran the local and global training computations inside SGX enclaves using SCONE - a shielded execution framework to enable unmodified applications to run inside SGX enclaves. In the SCONE platform, the source code of an application is recompiled against a modified standard C library (SCONE libc) to facilitate the execution of system calls. The address space of the application stays within an enclave. In our prototype, the input training data and code are encrypted using the file system shield of SCONE, and then decrypted and processed inside SGX enclaves which cannot be accessed even by strong attackers with root access. We rely on our previous works[20, 23] to implement the Configuration and Attestation Service.

2.5.4 AI-based optimization of Cloud/Edge Workflows

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.

On the other hand, traditional approaches apply heuristics to assign resources on demand without applying prediction techniques regarding future demands. While the approach allowed us 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 demands; thus, 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.

On top of this challenges, some of our use cases apply novel techniques for computational optimization using specialized resources such as GPUs or FPGAs. Such resources are limited in quantity, as they are pricey compared to computational power. Consequently, they are typically found only

on a subset of nodes. Moreover, the data producer may usually be located in other nodes. Therefore we find yet another challenge regarding data dispersion and resource heterogeneity that our orchestration policies must solve.

In the context of NEARDATA, we will explore the development of AI-enabled orchestrator that predicts future demands and requirements and assigns resources based on that knowledge. Thus, granting close-to-optimal allocations most of the time. Moreover, our orchestrator will be further improved by applying specific AI-based algorithms to consider the heterogeneity of resources, considering performance boosts via using specialized resources. This performance boost will be analyzed not only in terms of computational time but as well 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. On the other hand, extreme-health use case data comes from many locations (hospitals, laboratories) at once. Traditionally that data is moved to centralized data centers where the computational power lies for faster processing. The time spent on data transfer is typically not considered when reporting performance. In NEARDATA, we propose minimizing data movement by applying novel techniques such as hardware disaggregation [24, 25], which allows virtually possessing specialized resources on any node simultaneously. Applying this technique allows us to compute the data where it is, i.e., in the edge nodes (hospitals or laboratories). This can reduce the computational time spent on data transfer. While this solution can achieve some gains, there are many occasions when there is no alternative to using a cloud provider. Thus, for those situations, and specifically in the serverless context, we will make use of the Floki framework [26], which enables direct communication between functions in our use cases and significantly improve performance by reducing time spent by moving data in and out from the shared object storage.

Our initial proposal connectors to solve the aforementioned challenges are as follows:

- **Machine-learning training connector**: this connector will enable different machine-learning models. It will receive an input dataset and the selected model. As an output, it will deliver the trained model so inference can be applied.
- **Machine-learning inference connector**: this connector will apply the inference based on a given trained model and the input data.
- **Resource management connector**: this connector will provide several resource management policies. As an input will receive the workload requirements in terms of resources, as well as SLA requirements to be met. As an output will deliver the proposed selection for resource allocation.
- **Scheduling policy connector**: this connector will provide scheduling policies for the workloads. It will take as input the workload, the current situation of the system's resources, and SLA. As an output will provide the suggested resource allocation.

While initially, we plan to split the machine-learning connectors in this manner; we may unify both into a single machine-learning connection depending on the needs of use cases that might arise from the several modifications planned in the project. On the other hand, scheduling and resource management policies will be connectors used by our orchestrator. The connectors will suggest the resources and workloads; however, the orchestrator will ultimately decide where and when to assign the resources.

We propose to integrate this schema and policies on the Lithops framework as depicted in Figure 25. The figure shows the shadowed Lithops architecture on the background, and the new build architecture highlighted on the foreground. We can observe how, initially, the Lithops framework interacts with our resource management recommending system prior launching tasks, requesting predictions and configurations to be enforced. This is to ensure proper SLAs are met on all the workloads arriving into the data center.

Such recommending system is Machine Learning-based, and obtains data from the system using data-retrieval connectors providing telemetry metrics, while trains or adjusts statistical learning

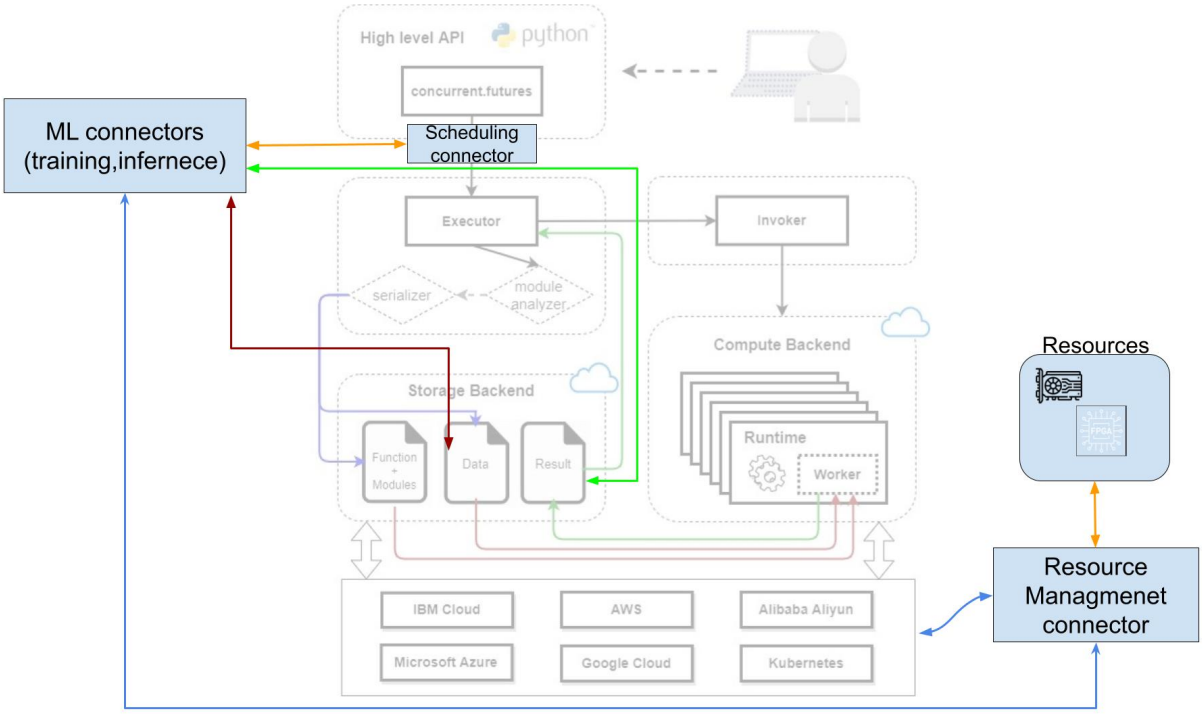


Figure 25: AI-enabled Lithops framework

models, and returns recommendations on placement, quotas or scheduling through a data-prediction connector. Once workloads are in a running stage, telemetry will be fed to the modeler/predictor as a ML-data connector, fitting a model for the workload behavior, characterizing its properties towards predicting or forecasting future behaviors. The obtained model will be available for the modules in charge of provisioning resources and placing workloads, asking for specific predictions towards deciding the best possible resource allocation and workload placement. Following the classic machine learning strategies for model maintenance, results from placement and provisioning can be collected to refine or reinforce the machine learning models, also evaluate and retrain them.

For such architectures, current technologies developed at the Barcelona Supercomputing Center, in collaboration with IBM, focus on different techniques for learning and planning from executed workloads. As an example, AI4DL [27] is a characterization algorithm for multi-variate time-series (e.g., telemetry traces for CPU, memory and I/O), used to discover workload behaviors along time. The algorithm uses a an encoder (i.e., a Conditional Restricted Boltzmann Machine) to embed the multiple dimensions plus time into a reversible hashing code, maintaining similarities between codes generated from similarly behaved time-series. When maintaining the similarity relation, such hashing codes can be clustered using traditional clustering methods (i.e., k-means). In addition, the method can be used for time-series forecasting. Therefore, time-series can be characterized, forecasted and compared (clustered) for similarity detection. The obtained clustering model determines the number of behaviors that can be detected, and from each we can extract the statistical characteristics per variable in the time-series. Such information can be leveraged by resource provisioning policies towards expecting sudden behavior changes or behaviors with high variance (like sudden bursts). Figure 26 shows an example of a trace with different behaviors automatically detected and clustered by similarity.

Aside of AI4DL, at this time we are researching different aspects of workload characterization with respect resource management:

- New methods for characterization that involve complex behavior patterns, with less interaction from human operators (remove the human in the loop), with less computational costs on training and predictions, and with better generalization and accuracy for models.

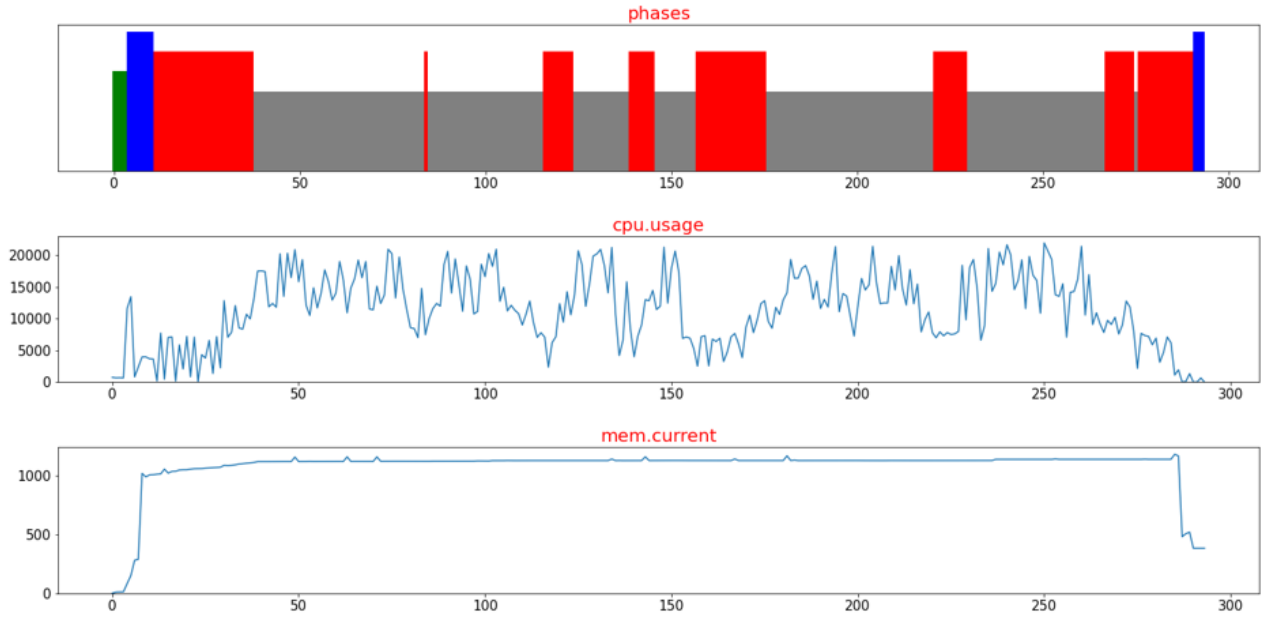


Figure 26: Automatically detected behaviors across a workload using CPU and Memory usage, including warm-up (green), low resources demand (blue), high variation on demand (red), and high demand (gray)

- How to apply characterization information (AI4DL and other models) into policy enforcement, for resource provisioning and application placement. This involves how to design and deploy data connectors that link "smart" components making predictions and decisions with the components controlling the computing environment.

For this, NearData will also research on the sub-specification data-connectors that should be designed to integrate these learning components in the proposed data-spaces environment.

3 Description of use case scenarios

As mentioned above, NEARDATA is based entirely on problems specific to healthcare domains. In the following section, we will detail each of the use cases presented by the different partners on different healthcare topics. The description and objectives of each one will be presented along with the technical requirements to carry them out, such as the data types, datasets and the data connectors. Finally, the experiments to be performed will be presented along with the integrations with the different components presented above.

3.1 Clinical sequencing of human pathogens

3.1.1 Description of the use case

UKHSA is the UK Health Security Agency, the government body charged with taking care of public health in the UK. As such, it routinely sequences a number of different human pathogens, both viral and bacterial, in order to produce clinical reports and respond to outbreaks. The number of pathogens sequenced can be estimated in the range of 10,000-20,000 each week, or 500,000-1,000,000 year. Using a conservative estimate whereby the sequencing of each pathogen generates 1 GB of data, that would amount to 0.5-1 PB/year just for the input data, which should then be multiplied by a relevant factor if intermediates and results of the analysis are considered. This scenario is complicated by the following requirements:

- The patient metadata (name, age, address, clinical history, NHS number, etc.) contains personally identifiable information. Under the reasonable assumption that the computational environment is not secure, such information must be removed from the data processing flow at the beginning of analysis and added back to it in the end, when the results of the analysis are uploaded to secure servers accessible to clinicians.

- The computational workflow needed for each pathogen is in principle different, as it depends on its genomics and its population structure and dynamics. That translates into the need for a modular library of analysis components, that UKHSA will then assemble as needed in order to obtain a suitable and optimized workflow for each pathogen.

- The computational requirements for the analysis of each pathogen are typically very diverse, depending on both the pathogen and the analysis stage. The first stages are usually based on read alignment and assembly; they are CPU- and/or memory-intensive operations, and require computational platforms with high data throughput. The following stages are usually less computationally demanding and based mostly on databases, communication and visualization, with a number of exceptions depending again on the pathogen.

- The generation and querying of samples for clinical analysis happens across a number of public bodies (hospitals, regional health agencies), which makes bespoke data processing and federation paramount. While some public bodies might perform their analysis on dedicated HPC, others might do so in the cloud. In addition, due to the constitutional structure of the UK as a union of four different countries, different requirements and rules might apply to clinical data depending on the place it is generated or used.

- There might be demand peaks, for instance when a pandemic such as COVID-19 happens, or during an outbreak generated by a specific pathogen. There might also be almost-real-time constraints, whereby the analysis of a sample needs to be performed within a fixed and short amount of time in order to comply with the requirements of clinical practice.

The pathogen-specific requirements are very well suited to an approach based on data spaces and connectors. In addition, one would be interested in a flexible data architecture that is distributed in nature and can easily be re-optimized on the fly or on demand. Due to an intrinsically federated system and the presence of privacy and access control issues, the solutions for the proposed use case should also include built-in data security (such as encryption of personal identifiable information)

coupled with appropriate access points and procedures ensuring that data and results are distributed correctly.

3.1.2 Datatypes and datasets

The data used in this case consists of two main categories:

- **Patient metadata:** Due to privacy and data security issues, patient metadata is usually not uploaded to global databases, with the notable exception of a small number of studies which have received ethical and patient approval and consent. In order to cope with issues related with personally identifiable information (PII), in this project we will simulate metadata based on the pathogen-specific statistics of the real data (age, geographical distribution, health provider, typical movement and patterns through the healthcare system). The use of realistic simulated metadata allows us to sidestep possible problems related with data protection rules, while at the same time retaining full scientific meaning and representativeness of the results.
- **Genomic data:** UKHSA routinely makes the data generated by pathogen sequencing publicly available on global databases for sequencing data such as the Short Read Archive (SRA) or the European Nucleotide Archive (ENA). Currently there are roughly 100,000 bacterial sequences that have been deposited in the SRA at URL https://www.ncbi.nlm.nih.gov/bioproject/248064. We will use an adequate subset of this data as test set during the set-up and scaling-up phases, focusing on at least two pathogens of interest to UKHSA.

3.1.3 Data connectors

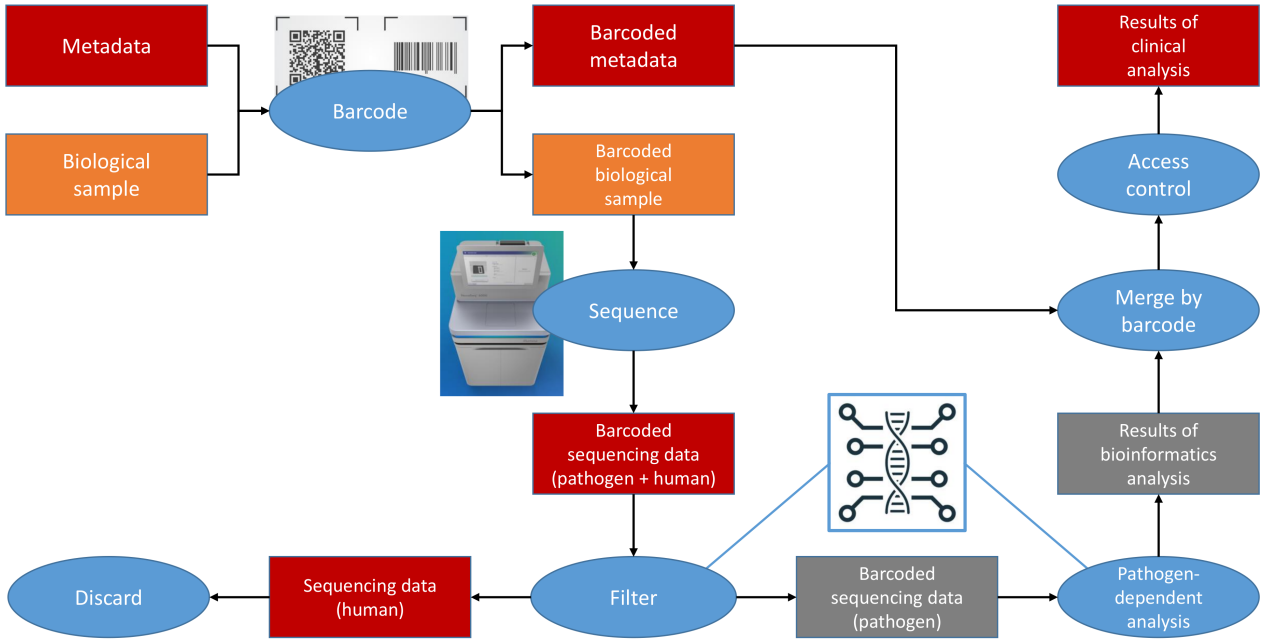


Figure 27: Overview of data flow in the clinical sequencing of human pathogens. Sensitive data in red. Note that the sequencing of human material in biological samples can also lead to the patient being identified even in the absence of patient metadata, due to the presence of recognisable variation fingerprints that can be searched for in available ancestry databases.

Our use case is depicted in detail in Figure 27. It can be conceptually divided into a number of data connectors, both for the handling of patient metadata and genomic data:

- **Metadata encrypter.** It encrypts sensitive data in such a way that operators with a sufficient access level (which might be different depending on whether the operator belongs to a regional or national body) can subsequently decrypt it.

- **Data-metadata unlinker.** Sensitive patient information must be separated from pathogen sequencing data, which is not personally identifiable, prior to analysis of the latter. However, one must be able to re-link the two later on.
- **Data analyser (sequencing data quality controller).** It performs quality control on sequencing data, and provides a basic description of the contents of the sample by querying a sequence database. Can be pathogen-dependent.
- **Data analyser (sequencing data assembler).** It performs de-novo assembly of sequencing data.
- **Data analyser (sequencing data aligner).** It aligns sequencing data to a known reference genome for the pathogen.
- **Data analyser (pathogen variant caller).** It identifies variants (for instance, Single Nucleotide Polymorphisms) from aligned sequenced data.
- **Data analyser (pathogen typer).** It uses either a de-novo assembly and/or variants in order to classify which sub-species, lineage or strain the pathogen belongs to.
- **Data accessor (pathogen database).** It identifies in a pre-existing database pathogens similar to the one(s) present in the sample being analysed.
- **Data comparator (relations among pathogens).** Given a number of pathogens, either present in a sample or identified in a database, establishes and display relations among them, for instance by means of a phylogenetic or non-phylogenetic tree.
- **Data-metadata linker.** Inverts the operation performed by Data-metadata unlinker.
- **Data merger and report generator.** Given the results of sequencing and analysis and a set of similar sequences retrieved from a specific pathogen database, it generates a human-readable report about the sample.
- **Metadata decrypter.** Inverts the operation performed by Metadata encrypter, taking data access levels into account.
- **Data slicer and accessor (clinicians, public health bodies, decision makers).** It implements data access levels for the full report (other components of the results/report might need selective access in addition to personally identifiable metadata).

3.1.4 Experiments

UKHSA has already implemented, and routinely performs, sequencing-based surveillance and testing for a number of pathogens. However, there is no integrated vision of the process, with many of the connection tasks between different stages (such as the barcode-based reconnection of metadata and sequencing data) often initiated by hand. Within NEARDATA, we plan to achieve two main goals:

- **Orchestration of the federated environment by stream processing.** This task focuses on the re-definition in terms of data streams of all the overall processing steps identified in Figure 27. With a technology such as PRAVEGA, data flowing between the different steps will be turned into streams; suitable events will be emitted whenever data moves from one stage to the other and transmitted to servers, allowing for a continuous and fine-grained control of the process. Notably, this architecture will allow a precise understanding of data flows, access permissions and policies for all the actors involved in the tasks – clinicians, hospitals, analysis laboratories, sequencing centres, public health agencies, epidemiologists and decision makers in the government and elsewhere.

- **Porting of existing pathogen-specific pipelines to data connectors and, possibly, confidential computing.** This tasks focuses on the re-definition and re-implementation in terms of the data spaces and connectors identified above of at least two pathogen-dependent analysis workflows currently used at UKHSA and collaborating institutions (typically *Salmonella enterica* and *Escherichia coli*). This will allow the creation of a modular library of components that can be extended to other pathogens of interest in the future. Also, this redefinition of workflows will help optimise the computational resources needed to process data. In order to protect patient confidentiality better, in particular regarding potentially identifiable human genomic data, we will also explore the possibility of performing part of the workflow on platforms for confidential computing such as SCONE.

3.1.5 Technical challenges

Technical challenges are mostly given by:

- The complexity of the process, which involves the interaction of a number of diverse actors who operate from many geographically separated locations. This will require a precise definition of both use case statistics and the process.
- The inherent large-scale of data analysis, which involves the processing of tens of thousands of samples per pathogen per year. It is unclear whether technologies such as PRAVEGA or SCONE will be able to cope with such large amounts of data in their current implementations.

3.2 Variants interaction analyses in massive genomics datasets

3.2.1 Description of the use case

The human genome reference sequence is composed of a sequence of >3 billion base pairs. By comparing the genomic sequences between any two individuals from the population, 3.78 million of differences (Genomic Variants) can be found. In particular, there are more than 400 million known variants. Among others, the study of genomic variation is crucial to understand disease predisposition and, therefore, one of the main goals of the Computational Genomics field is to identify disease-associated variants.

Complex diseases, such as Type 2 Diabetes, asthma, or Alzheimer's disease, are caused by the simultaneous effect of multiple genomic variants and other environmental factors. Thus, suggesting the relevance of the study of variant interactions and its effect on the risk of developing the disease. The effect of these variant interactions can be additive or synergic (epistatic). Since the study of disease-variant associations has been broadly approached, in a single independent manner, by Genome-Wide Association Studies (GWAS), one of the common ways to inspect the additive model is to directly measure the contribution of the sum of variant effects to disease development. However, this type of approach, which rely in previous knowledge, does not allow the identification of groups of variants that simultaneously, both in an additive or epistatic manner, contribute to the development of the disease. Additionally, the vast number of combinations (>6x10e13) needed to analyse pairwise interactions in a regular-size dataset (>10 million variants), converts the study of variant interactions into a still challenging problem that can only be approached using HPC technologies.

To approach these genomic and computational challenging problems we have designed machine learning and statistical frameworks. Within the NEARDATA project we aim to standardise these methodologies and to find computational solutions in collaboration with other groups. Therefore, this use case will contribute to the creation of new data connectors which can be used by the Computational Genomics community to analyse multiple complex diseases.

In order to approach the analysis of variants interaction in a real dataset, we present two different approaches: 1) using machine learning to discover of groups of variant interactions associated with Type 2 Diabetes at a Genome-Wide level (GWD), and 2) develop of a Multifactor Dimensionality Reduction method to enhance the discovery of pairwise variant interactions associated with Type 2 Diabetes (MDR).

- **Machine Learning approach, GWD:** machine learning classifiers have been proved useful in the identification of the most relevant variables to classify patients in groups of diseased and non-diseased individuals. Particularly, at the Barcelona Supercomputing Center, we have been able to apply these methods to find groups of genomic variants associated with the risk of developing Type 2 Diabetes. However, these types of approaches only allow the inspection of a few variables based on the number of observations. More specifically, the number of variables analysed cannot exceed the 10% the number of observations. For this reason, with extreme attention, we have only been able to analyse a dataset with 105,896 variants and >22 thousand individuals (pilot dataset). Within the NEARDATA project, our aim is to improve this methodology, to optimise its performance, and to use it in a combinatorial way to allow the inspection of >15 million variants, which is a still extreme computational challenging problem.

  To validate the diverse strategies and connectors developed within the NEARDATA project, two benchmarking cases will be used: 1) synthetic data and 2) pilot dataset. For the first case, we will generate the genotype and phenotypes of thousands of individuals from the European population, including some variant interactions. Both datasets, the synthetic and the pilot dataset, will be analyzed using the newly adapted machine learning algorithms to ensure the stability of the codes and the improvement of the performance. The results obtained will be compared with the provided and already working previous methods.

- **Multidimensionality Reduction approach:** in this approach we want to analyze every possible pairwise combination, meaning that we are dealing with more than 6x10e13 combinations per cohort. To do so, we are using Multifactor Dimensionality Reduction (MDR), a statistical approach for detecting a combination of variants. MDR is based on contingency tables as well as Chi-Square. It reduces the dimension of the problem. More specifically, it converts the counts obtained for the cases and controls into a simple binary variable by classifying all the possible allelic combinations for each pair in high-risk/low-risk, reducing the analysis to only one dimension. It follows a naive Bayes approach, building a probabilistic classifier from every variant-variant interaction and summarizing the best combinations for prediction. It can be implemented in 5 different steps, illustrated in 28.

  While MDR has been proven to be a successful method for detecting interaction, due to the dimension of the problem it must be implemented with some previous data selection step, reducing the dimension of the input. However, using the right HPC tools it should be possible to create an environment where processing a whole genomic cohort becomes feasible.

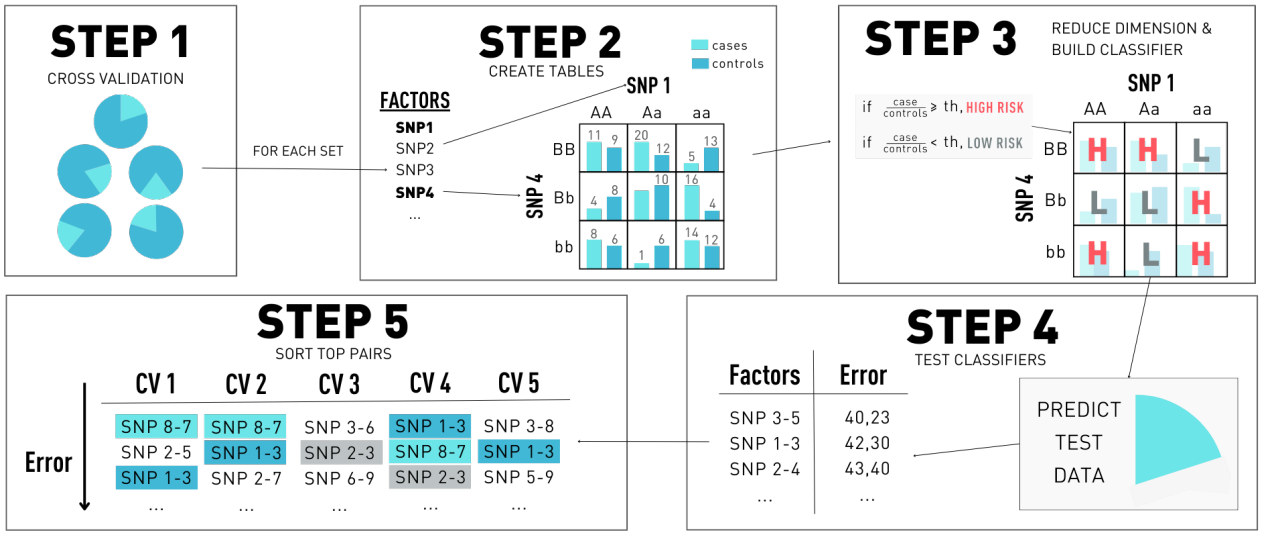


Figure 28: The 5 steps performed in the Multifactor Dimensionality reduction.

3.2.2 Datatypes and Datasets

The data used in this use case consists of the genotype and the phenotype of the individuals from 1) a synthetic dataset and 2) the 70KforT2D.

- **Synthetic dataset:** this dataset has been generated using EpiGEN and contains the synthetic genotype for 10,760 individuals of European ancestry in 101,327 genomic variants. It includes a pairwise interaction under a multiplicative model. This data will be used to validate and evaluate the diverse strategies and connectors developed within the NEARDATA project.
- **70KforT2D:** this a Type 2 Diabetes case-control dataset which includes data from 12,926 diabetic and 57,191 non-diabetic individuals of European ancestry. The genotype and phenotype of the individuals in this dataset is distributed in 5 previously published cohorts: Resource for Genetic Epidemiology Research on Aging (GERA), Finland-United States Investigation of NIDDM Genetics (FUSION), Wellcome Trust Case Control Consortium (WTCCC), Gene Environment Association Studies initiative (GENEVA), Northwestern University NUgene project (NUgene). The genetic information can be accessed through the dbGaP platform for FUSION (phs000867.v1.p1), GENEVA (phs000091.v2.p1), NUgene (phs000237.v1.p1), GERA (phs000788.v2.p3), and the Sanger platform for WTCCC. It includes information of 15,131,345 variants. Part of this dataset (pilot dataset) will be used to evaluate the strategies and connectors developed within this project.

3.2.3 Data connectors

Our use case can be divided into 4 main data connectors: 1) Data preparation, 2) Data selection, 3) Test of association, and 4) functional interpretation.

- **Data preparation:** in this first main connector we group all the connectors related to the initial load of the data, identifying three connectors: data ingestion, data clean, and data store. The purpose of this block is to prepare the data into a fast and easy-access database, that can be distributed.
- **Data selection:** in this second, we group all the connectors related to the access to the data. We have identified also three connectors: data extraction, data transform, and data merge. The purpose of this second block is fast access to the information of the database that every process may need.
- **Test of association:** in this third connector we apply the main algorithm to the data.
- **Functional interpretation:** in the last connector we study the results obtained, looking for a functional interpretation of the variants obtained from a biological point of view.

3.2.4 Experiments

While both two approaches for variant interaction are already developed, they are both limited by the extreme data needed to be processed. Therefore, we need to accelerate the processes, implementing better environments to reduce the computational time or reducing the computational cost making the connectors more efficient.

- **Serverless implementation:** servleress aims to reduce the costs for the allocated resources in the cloud. This is achieved by splitting the workloads into small tasks, called functions. Those functions get assigned the requested resources only for the period of time during which the function runs, hence, paying the price for the resource only for the effective usage. We aim to reimplement our use-case using functions for the different tasks and optimize the use of resources in terms of operational costs. To achieve this we will use Lithops as our serverless framework.

- **Serverless Data Connector:** fast access to the data is one of the key points of managing Extreme Data problems. One of the challenges faced is data partitioning according to the computational nodes consuming it. While static partitioning allows efficient division of data across nodes, data needs to be re-partitioned when those nodes change in characteristics. When having large datasets, such a process is time-consuming. Implementing the Serverless Data Connector to avoid creating static partitions and using a data virtualization model that enables effective partitioning to consume directly from Object Storage can be a great option to reduce the computational expense of the architecture. Data plug main benefit is that this partitioning process is made faster than traditional approaches.

3.2.5 Technical challenges

As introduced earlier, analyzing all the possible combinations that happen in a genomic dataset is a hard task, computationally speaking. The volume of data that we have after making all the pairwise combinations convert our problem in an Extreme Data situation, and this is the simplest scenario. While different algorithms are being used for MDR and GWD, they face similar challenges and need similar solutions.

- **Data management plan**: we have to read every variant several times in both our use cases. The amount of data that needs to be processed is too high to pre-load them on memory, so we need to be able to read every variant fast. Reducing the time needed to access and load them can lead to a great decrease in our total time needed to use both pipelines.
- **Method optimization**: even removing from the equation the reading time, the computational resources that both the MDR and the GWD need to process a whole genomic dataset is, nowadays, unfeasible. One of the main needs of both pipelines is to optimize the methods both reducing the computational time and improving the use of resources.

In summary, using the right technologies is key to get to analyze complete genomic datasets. Implementing a data management system that can deal with efficient access to the data and an HPC platform that allows a fast computation of the MDR and the GWD algorithm will increase the volume of genomic data we can study.

3.3 Transcriptomics Atlas Use Case

3.3.1 Description of the use case

Transcriptomic research, which focuses on studying gene expression patterns on a large scale, is a challenging field that holds great promise for understanding biological processes and diseases. However, researchers face several obstacles in conducting such research. One significant challenge involves acquiring an adequate number of patient samples for comprehensive analysis. Due to limitations in resources and time, researchers often find themselves struggling to gather a sufficient quantity of samples to draw meaningful conclusions. Additionally, the cost of sequencing has not yet reached a level where researchers can easily increase the number of collected samples within their limited budgets. To overcome this limitation, they often augment their dataset by including reference control group samples obtained from publicly available data repositories. These repositories contain transcriptomic data collected by previous research efforts, making them a valuable resource for current investigations. By incorporating these reference samples, researchers can expand their dataset without significantly increasing the cost of sequencing.

However, the process of incorporating publicly available data into transcriptomic research can be tedious and time-consuming. It involves several steps, including finding relevant studies using platforms such as PubMed [28], isolating information on where the data is stored, and downloading and processing the data for incorporation into one's research. The more samples are found, the more comprehensive and robust the reference values will be, leading to more impactful research. Unfortunately, the process is currently mostly manual, requiring a considerable amount of time to find a suitable number of relevant studies.

Even if the process of finding relevant studies were automated, the sheer volume of data encountered can pose a challenge without access to suitable computational infrastructure. Modern DNA and RNA sequencing techniques generate massive amounts of data that require large-scale high-performance computing (HPC) or cloud computing resources. Providing transcriptomic researchers with access to such computational architecture is crucial in accelerating innovation in this domain.

Another important aspect related to transcriptomics is data privacy. Since the expression data is always linked to patients, handling such data raises privacy risks. Building confidential and secure data processing pipelines based on privacy-enhancing strategies is of utmost importance to protect the individuals the data is derived from and to comply with legal regulations.

3.3.2 Datatypes and Datasets

In our experiments, we will utilize publicly available data for analysis. Specifically, we will evaluate our transcriptomic data processing pipeline using the Sequence Read Archive (SRA) database [29]. The SRA database serves as a public repository that houses raw DNA sequencing data. It is meticulously managed and maintained by the National Center for Biotechnology Information (NCBI), a division of the United States National Library of Medicine. Researchers have the ability to upload, share, and access raw sequencing data generated from diverse high-throughput sequencing technologies, including next-generation sequencing. Since its inception in 2009, the SRA database has accumulated an impressive volume of approximately 12 petabytes of data. Researchers can access this valuable resource through multiple cloud service providers, as well as via the NCBI servers and command line tools. The archive boasts a comprehensive collection of data samples obtained from various studies, including:

- Raw sequencing data generated by high-throughput sequencing platforms.
- Quality scores, indicating confidence level associated with each base call.
- Metadata, including information about the experimental design, and attributes, such as organism, tissue type or disease condition.
- Derived data: associated alignment files, genomic variants, reference genome.

3.3.3 Data connectors

Within our use case, we anticipate several data connectors, that are needed to build a Transcriptomic Atlas and Federated Learning Pipelines:

- **Metadata extraction**: to obtain data samples belonging to specific tissue and cell groups, we will build a metadata processing system, which will allow us to process and classify metadata contained within SRA archive, to extract identifiers of the relevant samples.
- **Data downloader**: using the identifiers extracted using metadata information, we will download the relevant samples into a cloud computing environment before performing computations.
- **Expression quantification**: for each sample we will quantify expression values, as a first step of building reference expression atlas.
- **Data normalization**: the quantified expression values will be normalized, to build a robust gene expression reference.

Additionally, for the Federated Learning experiment, we will need the following data connectors:

- **Feature extraction**: the information contained within variant calling data has to be extracted, to include only features that are relevant for the machine learning model.
- **Feature encoding**: the selected information will be encoded to create numerical representation that can be used by the selected classifier.

3.3.4 Experiments

To streamline the process of obtaining samples for transcriptomic research, we will apply novel computing methods on a large scale to build a Transcriptomic Atlas Pipeline (Fig. 29). It will automate metadata analysis and offload large-scale data processing to the cloud environment, improving the researcher experience. The pipeline will find and process raw sequencing data, quantify expression values, and normalize them to build a robust control group reference for researchers. It will be implemented on a commercial cloud platform. Two operating modes are considered: predefined and on-demand. In the predefined case, the pipeline clusters raw sequencing data samples for specific tissues and builds a database of reference, normalized expression values. This data could be available through a web service for researchers to download and use. In the on-demand case, researchers trigger the pipeline after selecting constraints for the metadata analysis module to search for relevant samples. After the workflow finishes, researchers can access tailor-made reference expression values for their research.

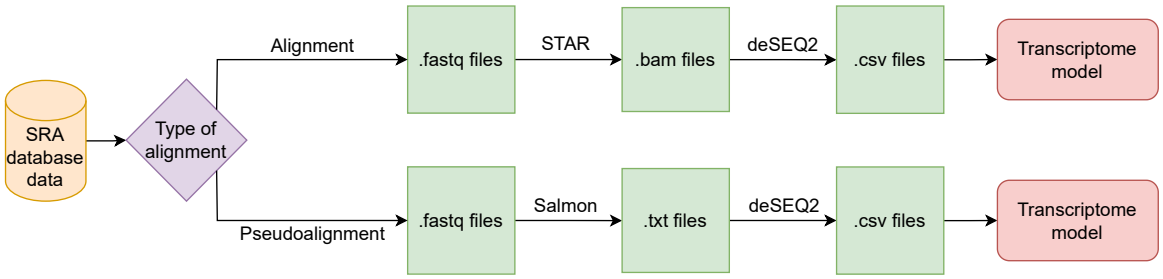


Figure 29: Transcriptomic Atlas Pipeline diagram, with file types at each step (.fastq/.bam/.csv). Depending on the alignment type, different tools can be used for data processing (STAR/deSEQ2/Salmon).

In addition to the Atlas Pipeline, our objective is to construct a Federated Learning Pipeline to explore distributed computing workflows in the transcriptomic domain. This pipeline will utilize data generated at one of the stages of the Atlas Pipeline to create a dataset containing genomic variant information. This dataset will then be employed in a federated learning workflow, where a network of distributed nodes such as organizations or hospitals will collectively train a consensus machine learning model for a classification task. By adopting this approach, we aim to evaluate the feasibility of applying federated learning techniques to train models using transcriptomic data. Moreover, this experiment will serve as a unique opportunity to highlight the advantages of federated learning, particularly in terms of enhanced data privacy resulting from the inherent no-data-exchange paradigm.

3.3.5 Technical challenges

The experiments contained within the transcriptomic use case combine technical challenges typical for a number of domains, such as Big Data (data processing, parallelization, scalability), Confidential Computing (data confidentiality) and Machine Learning (architecture search, data preprocessing, data representation). Specifically, depending on the experiment, the challenges include:

- *Transcriptomic Atlas*: efficient usage and processing of over 50TB of transcriptomic data despite high resource requirements (e.g. over 250GB RAM per instance for STAR aligner).
- *Transcriptomic Atlas*: parallelization of the alignment step to process subsets of data on different instances simultaneously, followed by merging outputs into single output file to improve the execution time of the pipeline for single input (SRA file).
- *Federated Learning Pipeline*: ensuring confidentiality of the processed data during every step of the federated workflow.
- *Federated Learning Pipeline*: scaling the federation to a greater number of organisations.

- *Federated Learning Pipeline*: efficient data preprocessing for large volume variant data.
- *Federated Learning Pipeline*: running large-scale federated learning experiments in cloud and HPC environments.
- *Federated Learning Pipeline*: architecture search for variant-based classification task.
- *Federated Learning Pipeline*: finding a suitable representation for genomic variants data.

3.4 Metabolomics Use Case

3.4.1 Description of the use case

Spatial metabolomics is a field of omics research focused on the detection and interpretation of metabolites, lipids, drugs, and other small molecules in the spatial context of cells, tissues, organs, and organisms. Spatial metabolomics is a rapidly emerging field, fueled by the strong and ever-growing need in biology and medicine to characterize biological phenomena in situ, as well as by the recently revealed key roles of metabolism in health and disease. This field is concerned with a variety of biomedical questions, including the tumor molecular microenvironment, functions of immune cells during homeostasis and immunotherapy, interactions between host and microbiota and their contribution to inflammation, regulation of early development, metabolic regulation of epigenetics, and metabolic dysregulations during infection and inflammation. Over the past decade, this growing interest has stimulated rapid progress in the development of enabling technologies – in particular, imaging mass spectrometry (MS) – that have achieved unprecedented sensitivity, coverage, and robustness as they have become accessible to biologists (Figure 30).

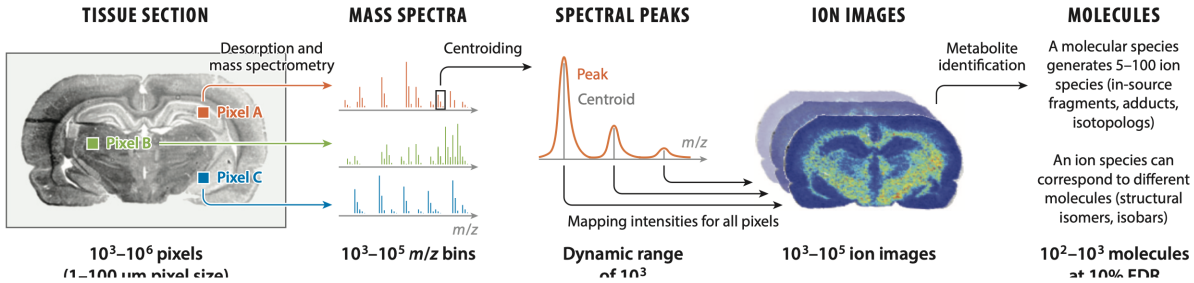


Figure 30: An imaging mass spectrometry (MS) dataset represents a collection of spectra acquired from a raster of pixels representing the surface of a tissue section. An ion image represents relative intensities of the ion across all pixels. An imaging MS dataset can represent spatial localization of up to 10^3 molecules.

We have developed METASPACE, a global community platform for spatial metabolomics populated by a large community of users [7]; see Figure 31. The cornerstone of METASPACE is a computational engine for metabolite annotation which searches for metabolites, lipids, and other small molecules in an imaging MS dataset. The engine estimates the False Discovery Rate (FDR) of metabolite annotations that provides quality control and – as demonstrated in other -omics – makes annotated spatial metabolomes comparable between datasets, experiments, and laboratories. We created a user-friendly web app for data submission and for interactive exploration of annotated metabolite images. By sharing their results publicly, METASPACE users cooperatively created and continuously populate a knowledge base of spatial metabolomes.

METASPACE was developed earlier in the European Horizon2020 project METASPACE (2015-2018). METASPACE integrates a high-performance cloud computing engine, a webapp for data submission, results search, browsing, analysis, and sharing, as well as a knowledgebase of private

[7] `https://metaspace2020.eu/`

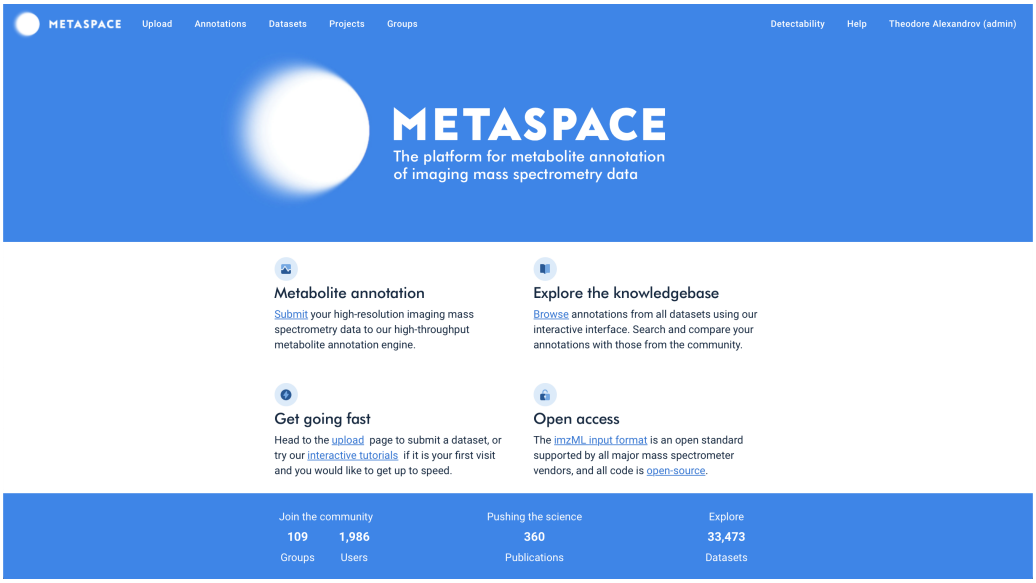


Figure 31: METASPACE landing page showing live statistics and providing access to webapp for data browsing, visualization, and metabolite annotation.

and public datasets and results from them. Since 2017, METASPACE became a major tool in spatial metabolomics with over 1900 registered users from over 100 labs, with many using it every day. We processed over 30K submissions, with over 1K submissions per month lately. Importantly, 25% of these submissions were shared publicly. This represents the largest public data collection in spatial metabolomics (and one of the largest in metabolomics in general) and, with provided metadata, a continuously-populated knowledgebase of spatial metabolomes.

Importantly, METASPACE requires scalable computing, taking into account the growth of the field (Figure 32) as well as the diversity of the datasets submitted to METASPACE in its nature and size.

3.4.2 Datatypes and Datasets

In our experiments, we will utilize data publicly available at METASPACE. Specifically, we will evaluate the results of the Experiments 1-2 using over 8000 public datasets [8] as well as EMBL private datasets. The total size is over 10 TB and the list of public datasets is updated multiple times a day.

METASPACE space is developed, managed, and maintained by the Alexandrov team at EMBL (the partner in this project). METASPACE users submit the datasets following the METASPACE Terms and Conditions [9] to use METASPACE services in particular the metabolite annotation. Since the inception in 2014, METASPACE accumulated over 30 TB of data, with the half of the whole data volume submitted in the last year. METASPACE users can access this resource through a web browser as well as programmatically through Python API. METASPACE hosts the largest collection of spatial metabolomics datasets. METASPACE stores all datasets in the centroided imzML format [10]. The format can be accessed using Python by using our package pyimzML [11]. The datasets contain information about spatial metabolomes from various organisms, organs, and analyzed using various spatial metabolomics technologies [12].

3.4.3 Data connectors

We anticipate the following data connectors to be necessary:

[8] https://metaspace2020.eu/datasets
[9] https://metaspace2020.eu/terms
[10] http://imzml.org
[11] https://github.com/alexandrovteam/pyimzML
[12] https://metaspace2020.eu/datasets/summary

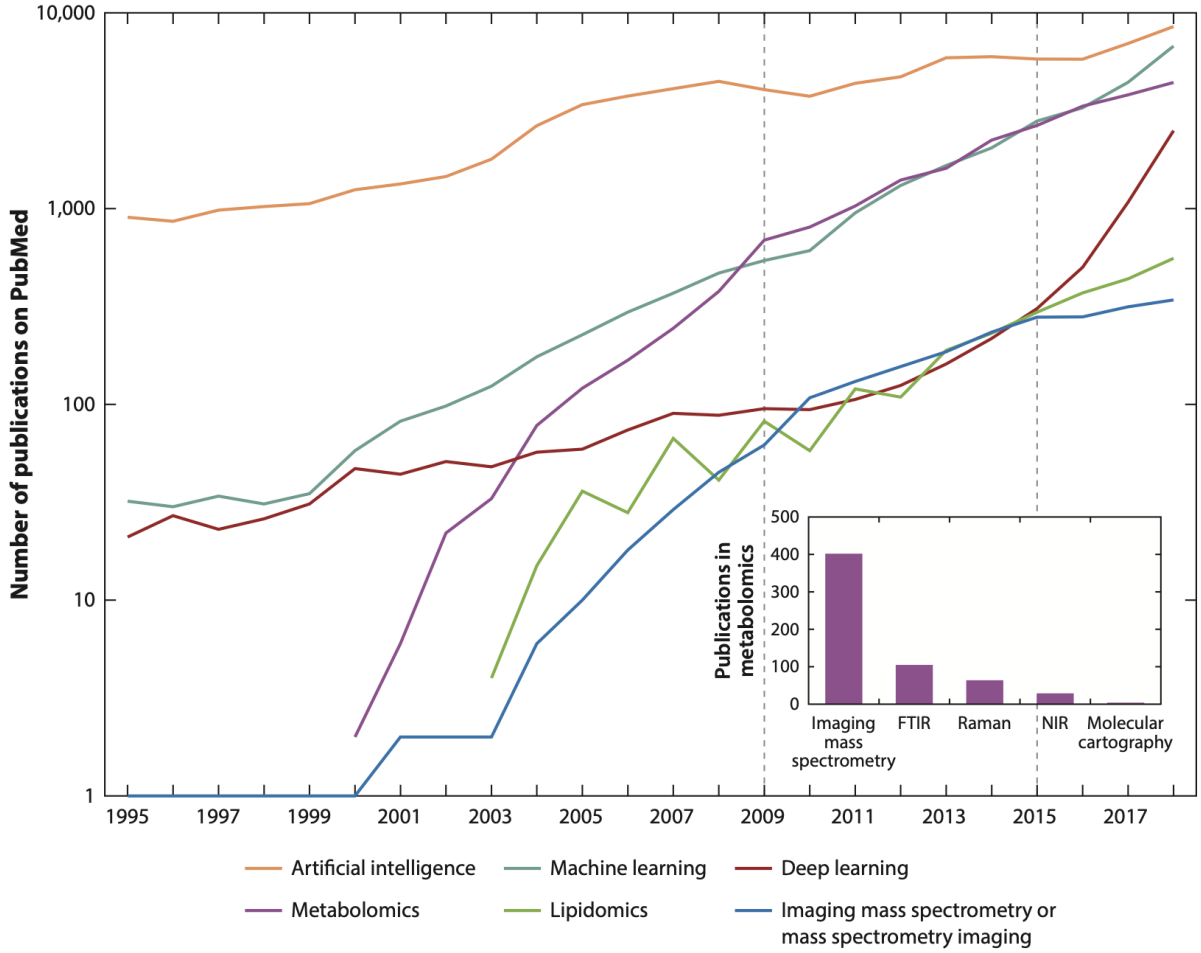


Figure 32: The popularity of different technologies in the life sciences and biomedicine and their evolution over time. The plot shows the numbers of PubMed-indexed publications in a given year containing the keywords shown in the figure key. We highlight three time periods, before 2009, from 2009 until 2015, and after 2015, which we discuss in the main text. The inset shows the popularity of several technologies for metabolomics applications from 1995 until 2018.

- **Metadata extraction**: to query metadata for datasets stored on METASPACE e.g. the types of mass spectrometry used, or the organ analyzed.
- **Data downloader**: using the identifiers obtained either from metadata or supplied externally, we will download the relevant datasets into a cloud computing environment before performing computations.
- **Feature calculation**: for a dataset and ion, we will calculate features quantifying the likelihood of the ion to be present in the dataset. This will include the features proposed by us earlier (e.g. rho_{chaos}) as well as novel features quantifying mass accuracy.
- **False Discovery Rate calculation**: we will implement efficient FDR (False Discovery Rate) calculation based on the target-decoy approach. This will require the generation of the target and decoy ions, calculation machine-learning-based scores for them, and ranking them based on the score, and calculating FDR values for each ion.

3.4.4 Experiments

To carry out the metabolomics use case we present two different experiments:

- **Mining historic METASPACE datasets with the recently developed ML model to address the "dark matter" problem in spatial metabolomics**. Recently, we have developed a novel method for metabolite annotation in spatial metabolomics, based on using machine learning (Wadie, Stuart et al, BioRxiv 2023). This method can find more metabolites in the spatial metabolomics data compared to the currently implemented method (Palmer et al, Nature Methods 2017). The development of this method can help address the so-called "dark matter" problem, namely finding molecules in the currently un-annotated major part of data. In Experiment 1, first, EMBL will implement the ML-based annotation using the Lithops serverless computing. Lithops is a powerful framework for scalability and is already used in the production version of METASPACE. This will require implementing calculation of new features, as well as deploying the machine learning model (CatBoost) on AWS. Second, EMBL with the help of URV will develop a hybrid architecture that will allow off-loading ML-based annotation to the edge, either to the EMBL clusters or to the KIO testbed or on the European Open Science Cloud. The key aspects of this architecture will the execution of the Lithops workers on the edge supported by the object storage infrastructure located near the extreme data of the historic METASPACE datasets (the key details to be formulated in the first year of the project).
- **Confidential computing for safe offloading of ML-based annotation of "dark matter" for private datasets.** In Experiment 2, we will use confidential computing for safely performing ML-based annotation in external edge environment. For this, EMBL together with URV and Scontain will use the confidential computing framework by Scontain and its parts integrated into the NEARDATA data processing platform. Specifically, first we will formulate the requirements for the NEARDATA data processing platform to support the ML-based annotation in METASPACE. This will be done following the completion of Experiment 1. We expect this to require minimal custom modifications of the Scontain / NEARDATA platforms as we will use Lithops with minimal additional software requirements. Second, we will develop and review the confidentiality requirements and develop the design for confidential computing. Third, we will implement and stress-test the confidentially-executed ML-based annotation in the edge environment (either test-bed by KIO or the environment used in Experiment 1). Finally, we will process the private datasets from METASPACE and ingest the results into METASPACE database and ElasticSearch.

3.4.5 Technical challenges

In the Experiment 1 of NEARDATA, we propose to mine tens of thousands of historic METASPACE datasets by using the novel ML-based annotation. This is impossible because of the cost associated with the mining the extreme data accumulated in METASPACE by using the commercial clouds (AWS, IBM Cloud) currently used in the production version of METASPACE. To put into perspective of the Extreme health data, METASPACE currently harbors over 33.000 datasets, each of the size of 1-200 GB that sums to over over 30 TB data. The annotation of a dataset using the currently implemented annotation method costs from 0.1 to 10 euro. Our estimate for the costs of mining all historic datasets exceeds 30K€ and this is using the minimal parameters only. At the same time, using it with expanded parameters will likely increase the price by several folds. Importantly, such mining of historic datasets is foreseeable in the future when we will improve our machine learning annotation, so it requires a completely novel solution, such as the near-data processing platform to be developed in NEARDATA.

This experiment will build on Experiment 1, yet it will expand it by addressing the following problem. Currently, METASPACE harbors both public and private datasets. Under the METASPACE Terms and Conditions, the public datasets are available to everyone under the CC BY 4.0 license. However, private datasets are shared under the conditions of privacy. This is important for the reputation of METASPACE as well as for its uptake among potential industrial users. However, the annotation of private datasets in Experiment 1 in the edge environment poses a threat of the leakage of the content of private datasets and their metadata.

3.5 Surgery Use Case

3.5.1 Description of the use case

Increasingly powerful technological and digital developments in surgery provide a huge amount of valuable data which can be used to improve patient therapy. Although a lot of data is available, the numerous data sources are an overwhelming challenge for physicians, especially in the operating room. The aim of data-driven computer-guided (robotic) surgery (CGS) is to provide the surgeon with the right type of assistance at the right moment by turning the available data into useful information.

While laparoscopic surgeries are largely beneficial for patients compared to conventional surgeries, they are challenging for the surgeon due to a loss of depth perception and challenging hand-eye-coordination. One application of CGS is to work against this imbalance by providing the right assistance functions, such as providing the position of a tumour, predicting surgical complications, etc., by analysing the surgical workflow using the video stream from the endoscopic camera.

3.5.2 Datatypes und Datasets

While there are many types of data that can be acquired in the operating room, the most expressive data type, and the type which we will likely focus on exclusively, is video data. The records will be stored in large datasets with different video formats, such as:

- MPEG-4.
- MP4.
- AVI.

In the initial phase of the project, we will sporadically draw from the data that we have in-house to test and configure the systems we develop.

- Dresden Surgical Anatomy Dataset (DSAD) (University Hospital Dresden and NCT/DKFZ) [30].
- HeiChole (University Hospital Heidelberg and NCT/DKFZ) [31].
- Cholec80 (University of Strasbourg) [32].
- Synthetic video data (NCT/DKFZ) [32].

However, owing to our connection with the Surgical Facility on the university hospital campus, we ultimately hope to curate datasets which will highlight the use case in the context of this project specifically, by encompassing the variety of surgical workflows and consequently enabling the assessment of our developed infrastructure to handle the corresponding computational demands.

3.5.3 Data Connectors

This use case will develop data connectors designed for surgical tasks like: the detection and segmentation of organs and tools, determining the current surgical workflow phase, and registration of preoperative and intraoperative information. Broadly speaking, there are two central work themes which we have identified, and which our upcoming work will centre on:

- **Streaming-based distributed computing using GStreamer & Pravega**, with a focus on the implementation of algorithms serving Extreme Health Use Cases. This is principally in conjunction with Dell.
- **Confidential Computing**, specifically through Federated Learning approaches which leverage Secure Container Environments (SCONE) in the context of Machine Learning using data sourced from multiple healthcare centres. This is principally in conjunction with TUD.

With regard to the algorithms and systems that we will focus on implementing in the context of the project, they can be split into more independently-operating algorithms and those contributing to more complex interacting systems of algorithms (in surgical navigation, for example).

3.5.4 Experiments

Surgical phase recognition and instrument segmentation, for instance, will be taken as first candidates for implementing within the developed infrastructure as they can serve as stand-alone assistive tools. In either case, they are machine-learning-based algorithms.

Further down the line, once such independent algorithms have been successfully implemented, the more complex surgical navigation context will be addressed. This is more sophisticated as it entails the orchestration of several such independent algorithms:

- Liver Segmentation (Neural Network).
- Simultaneous Localisation and Mapping (SLAM, CPU-based).
- Disparity Estimation (Neural Network).
- 3D-Reconstruction System (CPU-based, leveraging outputs from the above 3 components).

Each of these serve a different role for the ultimate goal of illustrating in Augmented Reality (AR) the position of target structures (tumours and blood vessels) for the surgical team. The most appropriate components will be integrated into the NEARDATA framework.

3.5.5 Technical Challenges

One of the biggest opening challenges for the surgery use case is that this (stereo) video data has to be analysed in real-time with low latency. Depending on the surgery type and the progress of the surgery, the need for computation power can vary. Since the surgical data is difficult to share and use for machine-learning (ML) because of technological, ethical and legal concerns regarding patient and surgical staff privacy, data governance, anonymization, access control and secure data exchange must be employed in the handling of medical data. A major bottleneck for CGS is the requirement of multi-centric data with high variance to create generalised ML models that can aid surgeons during surgery, e.g. to avoid postoperative complications.

4 Benchmarking framework

In the following section the different requirements to perform the experiments and validations proposed above will be presented. The functional Software and Hardware requirements will be detailed, together with the environment in which it will be executed (Cloud, HPC) with the necessary services (Object Storage, Cloud Functions, ...). Finally, the Testbed offered by our partner KIO from all the requirements exposed during the section will be detailed.

4.1 Clinical sequencing of human pathogens

As mentioned before, we plan to conduct two main experiments Within NEARDATA. Their requirements are as follows:

- **Porting of existing pathogen-specific pipelines to data connectors and, possibly, confidential computing.** This tasks focuses on the re-definition and re-implementation in terms of the data spaces and connectors identified above of at least two pathogen-dependent analysis workflows currently used at UKHSA and collaborating institutions (typically *Salmonella enterica* and *Escherichia coli*). In detail:
  - We will identify a minimal modular subset of bioinformatics components that allow us to re-implement an arbitrarily large number of pathogen-dependent workflows. The general architecture in terms of data sources and connectors is the one identified in section 3.1.2; however, current UKHSA pipelines have been developed in an incremental way, with a lot of inconsistencies due to historical factors. A detailed description of use cases and technical requirements will help us reduce the set of needed tools to a strict minimum.
  - We will re-implement the subset identified at the previous point in a scalable way. In order to do that, we will use tools such as Lithops on AWS, which easily lends itself to elasticity, for instance in a serverless computing environment. We will build upon previous work already conducted within the previous project EU CloudButton, which led to the establishment of a highly scalable bioinformatics pipeline to perform variant calling on human data. Some of the bioinformatics components developed there have been integrated into Lithops, and we will rely on them and on this previous experience in order to get to the implementation of our minimum set of needed bioinformatics modules. This step will be tested both on the HPC and in the cloud.
  - We will explore the implementation of parts of the workflows (in particular, the ones implying the potential accidental manipulation of human genomic data, which can lead to identification of the patient through fingerprints deduced from genomic variants and found in existing databases) within a confidential computing framework. As a first step, we will explore how easily existing bioinformatics tools can be modified to be made them ready for confidential computing; as a second step, we will test the scalability of such solutions. We will do that with SCONE.
- **Orchestration of the federated environment by stream processing.** This task focuses on the re-definition in terms of data streams of all the overall processing steps identified in Figure 27. In particular:
  - We will identify all the phases of each workflow and associate them with data streams, in order to be able to make the state of each sample explicit and update it in real time. We will implement a client/server system, whereby the different data producers (clinicians, hospitals, analysis laboratories, sequencing centres) can use clients to update the system about their work on the sample, and workflow controllers (public health agencies, epidemiologists and decision makers in the government and elsewhere) can access the server in order to implement dashboards, management software and other tools.
  - We will also attempt to re-implement some of the minimum set of bioinformatics components identified above directly in PRAVEGA. That might involve, for instance, the splitting

of each sequencing dataset into a number of smaller sets of sequencing reads that can then be considered as a computational unit to be streamed and processed. It is unclear at the moment if this is possible with the current implementations of PRAVEGA, as the amount of data to be streamed and processed would be significant.

We will need PRAVEGA in order to achieve this goal.

4.1.1 Involved Tools and Systems

We will use the following Software tools and Systems in our experiments:

- Docker
- AWS
- Lithops (for serverless computing)
- Pravega (for distributed computing on data streams)
- SCONE (for confidential computing)
- Bioinformatics tools (a number of them, as needed after a suitable set of scalable modular components has been identified and implemented as described above).

4.2 Variants Interaction Analytics in massive genomics datasets

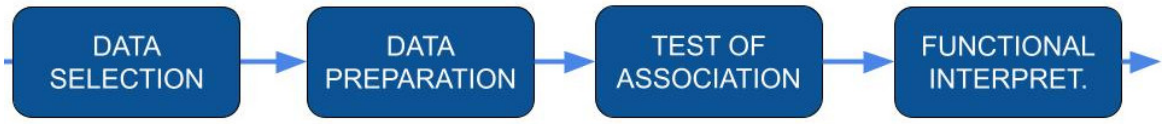


Figure 33: Variants Interaction Analytics pipeline stages

The Variants Interaction Interaction workload is divided into two different pipelines:

- Genome-wide level Discovery (GWD) of groups of variant interactions associated with Type 2 Diabetes.
- Multi Dimensionality Reduction (MDR) method to enhance the discovery of pairwise variant interactions associated with Type 2 Diabetes.

Both pipelines are structurally trying to solve the same problem by applying different solutions. Therefore both cases are an HPC problem to the extremely high volume of data generated: we need to compute all the variant interactions possible within the human genome, quickly becoming billions and trillions of combinations (more than 6×10^{13}) in a regular-sized dataset with more than 10 million variants. The current implementation attempts to, on the one hand, parallelize the computational part and on the other hand, limit the number of variations. Figure 33 depicts the main stages.

4.2.1 Cloud and edge computing environments

Within the NEARDATA project, we will explore further parallelization by applying a serverless methodology to parallelize the different pipeline stages, especially the data ingest which is one of the main challenges. Despite the problem being initially an HPC problem, each stage can be easily decoupled and divided into small tasks that will be translated into serverless functions, porting the use case into a cloud use case. Moreover, in the GWD pipeline, the XGBoost machine learning model is applied to the data. This data is currently obtained from various datasets (described in the previous section). However, these data sources were, at a point, hospitals or clinics. This delivers the opportunity to train the model in situ on several clinics where data can be collected, thus becoming an edge computing use case and enabling the opportunity to apply federated learning.

4.2.2 Pipeline evaluation

Summarizing this into experiments, we will evaluate the following elements:

- Data splits: in the data ingestion process, we need to seek how big the chunks partitioning should be according to the different contexts. This links to the Serverless Data Connector library, allowing us to adapt to the size of chunks we need.
- Accuracy of learning: when training the data into smaller chunks, we need to see which impact has on the overall accuracy of the workload with respect to the HPC pipeline, where all data is available at once.
- Data cleaning: patient data is often incomplete for several reasons, proving a challenge as our case analyzes variants of interactions among patients. Thus incomplete data may imply the impossibility of using the subject. A cleaning strategy must be devised to decide which data can be used in each situation.
- Data storage: depending on how the data is stored, the posterior computation becomes more manageable or harder. Thus, we analyze how different storage techniques impact use cases' performance. Moreover, we will study how we can leverage shared object storage enabled in NEARDATA.
- Data extraction: this stage extracts the relevant data. The challenge is how to extract significant data quickly. This is directly linked to the data storage stage.
- Data transformation: this stage seeks to filter data if possible. For instance, in the MDR pipeline, a heuristic is applied to minimize the number of variants that need to be computed. The filtered amount is later compared with a test dataset to find the accuracy of the pipeline.
- Data merge: finally, the different splits of data processed in parallel need to be merged into a single dataset. Different merging strategies should be analyzed to find the optimal one.

4.2.3 Involved Tools and Systems

Lithops and Serverless Data Connector will be our serverless frameworks and libraries in both pipelines.

- Software requirements: GWD pipeline only uses Python. While MDR uses Python, the Apache Spark runtime, PySpark, and Numpy libraries.
- Hardware requirements: in terms of hardware, the use cases do not require any specific hardware. However, in the context of NEARDATA, we will explore optimizations using specialized resources such as GPU or FPGA.

Given the use-case's main challenge is to process as many variants as fast as possible, both use cases are resolving HPC problems. Therefore one of the main benchmarking environments will be the BSC MareNostrum IV supercomputer.

Our pipelines will explore the efficiency serverless can provide using the Lithops framework. The first step will be porting the pipelines into the serverless environment. The experiments will be conducted on an OpenStack-enabled cloud environment.

For the connection with Serverless Data Connector we will need Object Storage. For this, we will use the shared object services provided on OpenStack and adapt them to behave as expected by the Serverless Data Connector library. On the other hand, to test the AI-enabled orchestration optimizer, we will enable Kubernetes in our OpenStack cluster.

4.3 Transcriptomics Atlas Use Case

4.3.1 Atlas Pipeline

To benchmark the applicability of serveless technology and confidential data exchange and orchestration in a transcriptomic setting, we will adapt an batch processing pipeline for the transcriptomic atlas and a federated workflow build on top of it. The original transcriptomics pipeline for the Transcriptomic Atlas use case consists of 4 phases:

1. *Download* SRR file using "prefetch" tool.
2. *Extract* fastq files using "fasterq-dump" tool.
3. *Quantify* expression using SALMON tool [33] or perform *Alignment* using STAR tool. [34][35]
4. *Normalize* counts using DESeq2 library in R.

Both "prefetch" and "fasterq-dump" are provided by "SRAtoolkit" [36].

Files generated on each step of the pipeline vary in size. The first step is to download the SRR file which are several GB in size, usually less than 10GB for our use case. In the next step, the space needed during the conversion to fastq is approximately 17 times the size of the accession and the resulting fastq files generated are about 7 times the size of the SRR file. The outputs of SALMON and R script are measured in MBs.

The goal is to create the Transcriptomics Atlas which requires processing tens of thousands of such SRR files. To make it comprehensive we will need to process files associated with about 20 different tissues. For each tissue, pipeline results from hundreds of SRR files are required. As stated above, the SRR files are usually several GB in size. With this specification we can estimate that the total size of all SRR files will exceed 50TB. Adding more tissues to the atlas in the future will increase this value.

The most time and resource consuming step in the pipeline is the quantification/alignment. For SALMON, the time to complete this step is dependant on an instance size and the SRR file size. For example an instance with 2 vCPUs, 8GB RAM and 7GB SRR file it can take a few hours to perform quantification. To perform alignment the STAR tool requires a very high memory instance (over 250GB RAM).

Originally the pipeline was created for HPC usage but we adapted each step for cloud usage. The current architecture is presented in Fig. 34. All the generated intermediate files (e.g. fastq) are not needed after the pipeline has finished. Also alternative approaches that would require splitting the pipeline steps among services incur heavy transfer and communications costs. Therefore each SRR file is processed on a single EC2 instance from start to finish of the pipeline. We are using Auto-Scaling Group in order to automatically scale number of instances based on SQS queue size. Final results are to be uploaded to an S3 bucket.

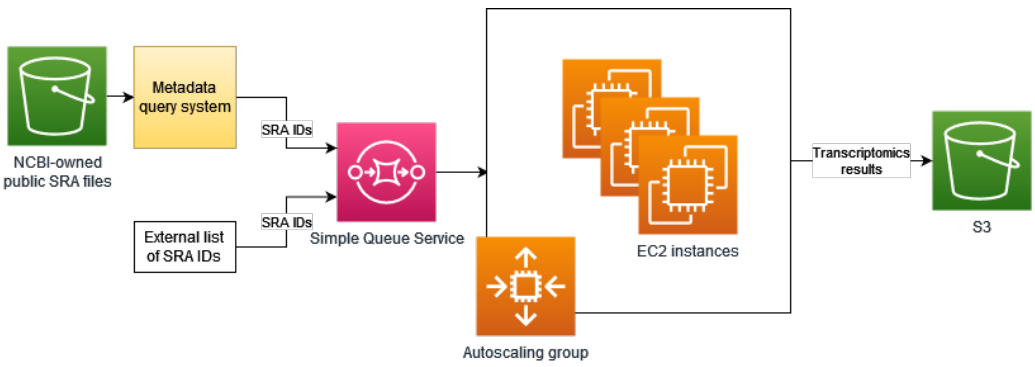


Figure 34: Transcriptomics atlas current architecture plan.

In order to integrate Lithops to the current pipeline we consider two architectures. In Fig. 35 we present possible architecture which utilizes Lithops API to AWS Batch. This service can run containers in serverless model using AWS Fargate launch model. With this integration with replace SQS service as well as replace boto3 library usage with Lithops in the main pipeline script (e.g. API calls to S3). With this change we are less vendor dependant and provide a way to execute containers in serverless manner. However due to Fargate memory limits [37] this pipeline is valid only for SALMON quantification and not for STAR alignment.

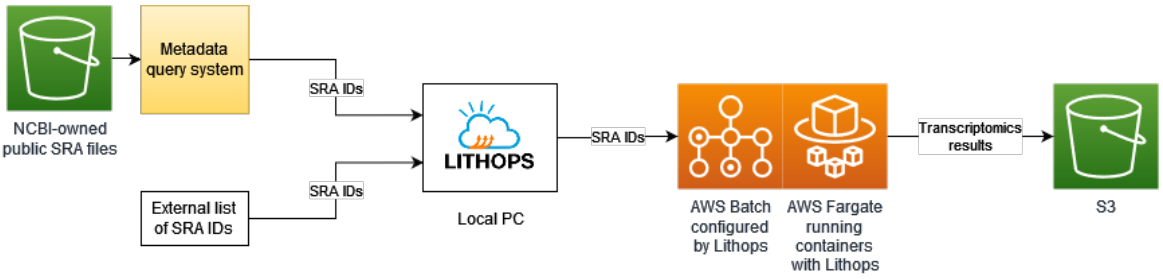


Figure 35: Transcriptomics atlas proposed architecture with Lithops and AWS Batch

Alternative architecture is presented in Fig. 36. Here we use Lithops API to manage EC2 instances. With this approach we create a solution available both for alignment and quantification using high memory EC2 instance types required by STAR aligner. In this architecture Lithops also replaces SQS service and provides auto-scaling of EC2 instances based on number of tasks (here SRR IDs).

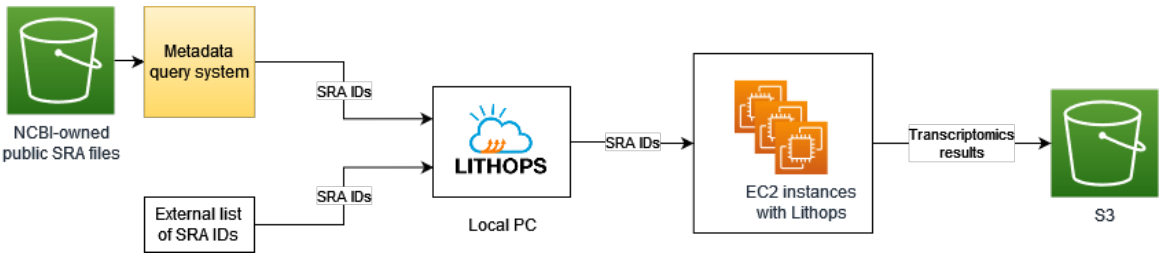


Figure 36: Transcriptomics atlas proposed architecture with Lithops and EC2

Integration of Serverless Data Connector in this architecture would allow to process parts of files and thus may reduce intermediate computational or storage requirements. This approach requires alignment algorithms that are able to process incomplete files. However, currently there are no such algorithms available. The final architecture and configuration will be influenced by resource requirements, results of the pipeline performance analysis and cost optimization.

4.3.2 Involved Tools and Systems

The tools and systems that will be used to create the Transcriptomics Atlas are listed below:

- Lithops
- SRA-Toolkit
- SALMON
- STAR
- R (with libraries e.g. DESeq2)

- AWS Services (EC2, S3, Batch, Fargate, SQS, CloudWatch)
- Python
- Docker

4.3.3 Federated Learning Pipeline

The transcriptomic data processing pipeline described above can also serve another purpose, apart from creating a transcriptome reference model. One can use the intermediary files (extracted *.fastq* files) from the second step of the pipeline to obtain *.vcf* files, which can be used for variant analysis. In the federated learning workflow for this use case, we will use the obtained variant calling format files as a data source for a federated learning model. In a federated learning workflow developed for HPC and cloud environment, we will divide the obtained variant calling dataset between participating nodes, to simulate a number of organisations (e.g. research centres/hospitals) forming a federation and train a consensus model.

An example federated learning workflow depends on the strategy used. Taking as an example a default FL strategy (Federated Averaging, [38]), we can distinguish several phases of building a consensus FL model:

- *Model initialization*: a model is initialized on a central server as a starting point for the training process.
- *Client selection*: a subset of nodes from the federation is selected to participate in a training round.
- *Model distribution*: the model is distributed to all selected clients.
- *Local model training*: using their own, local data, each client performs a number of local training steps of the model.
- *Model aggregation*: clients send the updated models to the central server, which aggregates them using weighted averaging.
- *Global model update*: the averaged model becomes the new global model. The training round concludes.
- *Convergence check*: after finishing the training round, model convergence can be evaluated on a test dataset to see if its performance matches training termination criteria. If not, the training round may be repeated starting from the client selection step.

The above federated learning workflow opens many ways to benchmark the data processing and confidential orchestration platform developed within the NEARDATA project. We intend to evaluate the use of the Lithops framework for parallel and serverless *.vcf* file processing. Data ingestion is an important part of the federated learning workflow. The data used to train the consensus model should be obtained by participating organisations, and for some participants it may be difficult to process large volume of variant analysis datasets, specifically when transforming raw data to a representation suitable for machine learning models. Before the data is ingested by a machine learning model, preprocessing steps have to be carried out, including feature extraction and feature encoding. These steps can be executed in a serveless fashion using the Lithops framework.

The serveless computation paradigm can be also used for orchestration purposes, as proposed in [39]. In this setting, the organizations participating in the federated training do not have to maintain dedicated servers for local model training at all times. Instead, the execution of this step can be done in a serverless fashion only when the respective organization is chosen to participate during the client selection step. Utilizing this computing paradigm can thus reduce operational costs and enhance the practicality of the solution.

Lastly, although the privacy of patient data exchanged is increased by training the consensus model in a federated learning workflow, where only model weights are transferred within the federation, the confidentiality of the computation carried out during local training steps is not enhanced in any way and remains the responsibility of the participating organizations. This means that an adversary can potentially attack the participants and gain access to sensitive information during local data processing steps, compromising data confidentiality and integrity. Therefore, running federated learning in Trusted Execution Environments (TEEs) can complement the data sharing guarantees with data processing security guarantees and improve the overall security of the system. In this regard, we will adapt the federated learning workflow to work with confidential data exchange and orchestration tools developed within the NEARDATA project by utilizing the SCONE toolchain for running local training steps.

4.4 Metabolomics Use Case

4.4.1 METASPACE pipeline

The project aims to improve the METASPACE platform for spatial metabolomics. Initially, METASPACE was implemented using the Apache Spark architecture. However, with the increasing yet highly variable numbers of datasets submitted per data, we started having increasing pressure in administering the queue. This increased demand in scalability was addressed in the European project CloudButton (2018-2022) where we have implemented the serverless version of METASPACE using the Lithops framework; see Figure 37. This allowed us to remove the need for administering the queue and the Apache Spark cluster and achieve the necessary scalability. The Lithops version of METASPACE was deployed on production in March 2022 and is the main version since then. Later in 2022, we deprecated the Apache Spark version completely.

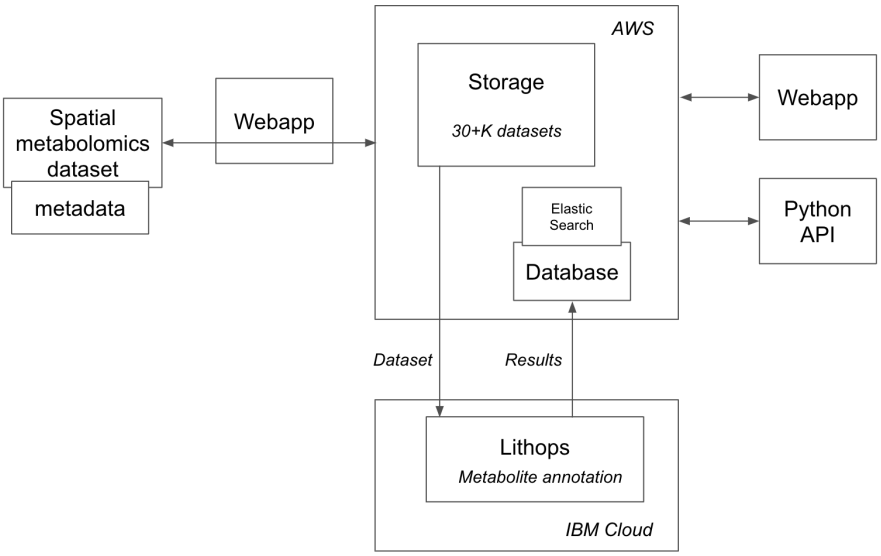


Figure 37: The current pipeline of the metabolite annotation on METASPACE using Lithops.

4.4.2 Experiment 1

In the proposed experiment 1 (Figure 38), we will mine the public datasets in METASPACE using a novel machine-learning version of metabolite annotation. This will be done by offloading the computations to either EMBL cluster, or to the KIO testbed, or on the European Open Science Cloud. The datasets will be moved there using Lithops and batach-processed using the ML-annotation. The results will be merged into the METASPACE database and will complement the currently available results.

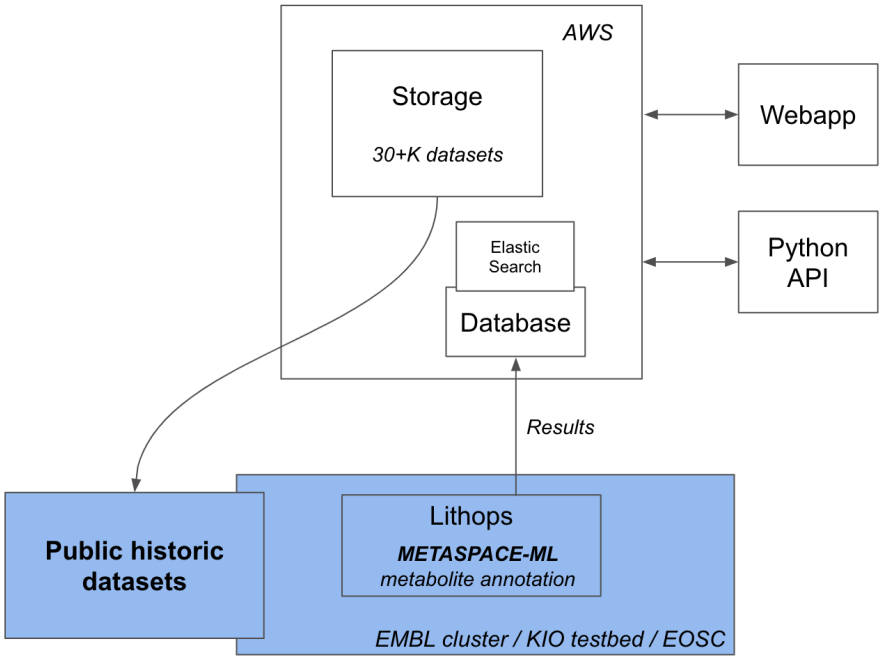


Figure 38: The changes in the METASPACE pipeline to be developed and implemented in the first experiment. The changes compared to the current pipeline are shown in blue.

4.4.3 Experiment 2

In the proposed experiment 2 (Figure 39), we will offload the re-mining of private historic METASPACE datasets (more than 30 thousand) onto an external computing resource similar to the experiment 1, but using the confidential computing. For this, the data will be encrypted and all computations will be performed in an encrypted enclave.

4.4.4 Involved Tools and Systems

We will use the following software for the Metabolomics use-case:

- Python
- Lithops (for serverless computing)
- AWS
- PostgreSQL
- ElasticSearch
- RabbitMQ
- Node.js

4.4.5 Evaluation and impact

The Experiment 1 is strongly aligned with the concept of NEARDATA as the Extreme near-data processing platform with offloading the compute-intensive parts to the edge whereas the results will be ingested into the current production version of METASPACE and provided to the users through the METASPACE webapp and API as usual.

The Experiment 2 is strongly aligned with the vision and concept of NEARDATA because the private datasets represent the largest part of METASPACE datasets (75%) and with over 25.000 datasets as of June 2023, of the size of over 20 TB. We expect this number to significantly increase during the

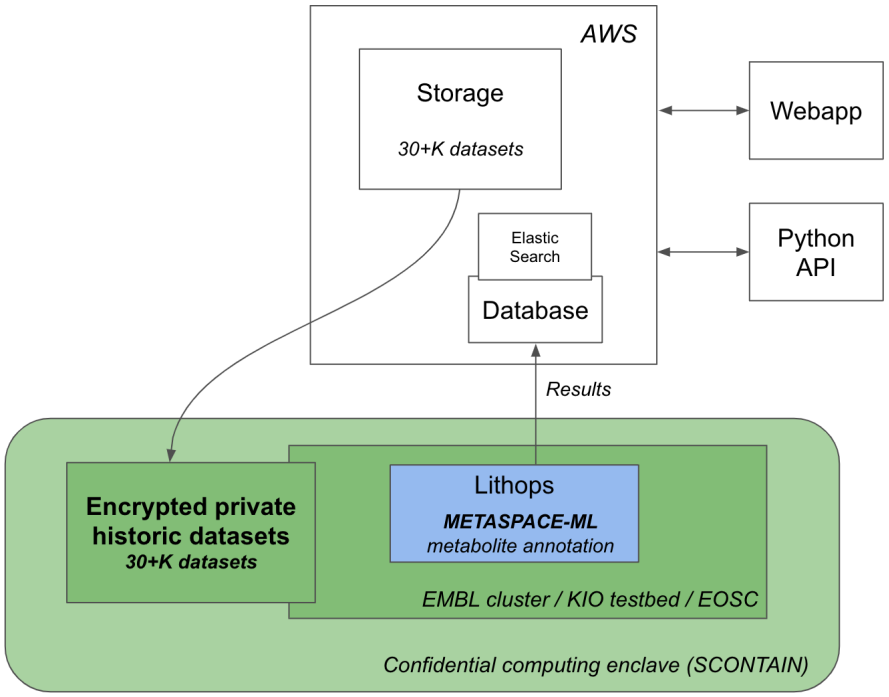


Figure 39: The changes in the METASPACE pipeline to be developed and implemented in the second experiment. The changes compared to the current pipeline are shown in blue and green; the changes compared to the first experiment are shown in green.

scope of the NEARDATA project as METASPACE is currently undergoing a super linear growth in the number of datasets and an exponential growth in the total volume of accumulated data. So, this experiment will address a growing extreme-data problem.

For benchmarking, we will use the following parameters to evaluate and benchmark: compute time, cost, data transfer time/cost, scalability, ease for developers, and ease of maintenance.

4.5 Surgery Use Case

In the NEARDATA project, two overarching experiments will be carried out in the surgery use case.

4.5.1 Surgical Navigation Pipeline

The project aims to develop a strategy for computer-assisted surgery using distributed computing. Robot Operating System (ROS) is the existing distributed computing solution within the NCT that will be used as a benchmark. The developed strategy will be validated using two benchmarking cases: surgical phase and scene recognition, and surgical navigation. In surgical phase and scene recognition, surgical videos will be analysed to provide assistance to the surgical staff. In surgical navigation, the most relevant components, such as liver segmentation and disparity estimation will be focused on for the downstream task of 3D reconstruction of the surgical scene, and subsequent visualisation of internal liver structures.

In such surgical interventions, the extreme nature of the data and its' processing requirements - high volume video data, requiring real-time analytics at the from Edge computing as well as long-term storage - gives rise to criteria which we will consider, such as Frames per Second (FPS), the latency of inferences in ms, or in Floating Point Operation per Second (FLOPS).

This experiment requires access to edge computing resources located in or near operating theatres and a larger computing infrastructure capable of storing the data generated by the experiment. Such a system can be built on the Pravega streaming storage system presented earlier. A simple example

architecture can be found in Figure 40.

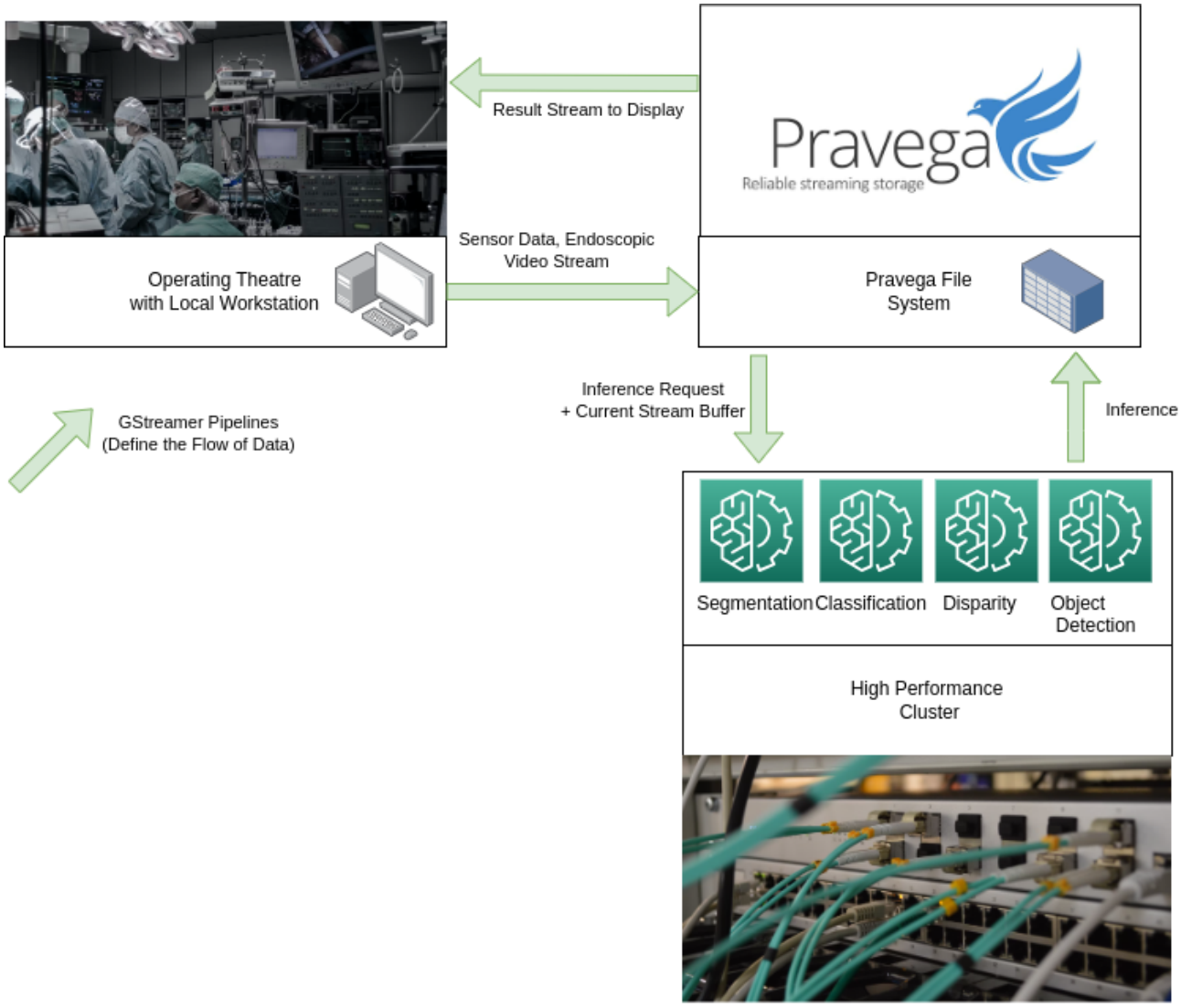


Figure 40: **The surgical benchmark architecture.** GStreamer pipelines (green arrows) are used to route data to and from Pravega. In a minimal example, the video stream from an endoscope is routed to Pravega via the local workstation in the operating room (OR). Another script is then run on the HPC to load the recently received data from the Pravega buffer and provide it as input to the appropriate neural network. The results of the neural network are then sent back to the buffer and to the server, or to the workstation in the OR so that the results can be displayed on a screen during the surgery.

4.5.2 Federated Learning Pipeline for Surgical Data

The second technical bottleneck we want to address in this project is how to train deep learning models on surgical data when different surgical centres are unable (or unwilling) to share their data, making this data extremely dispersed. One solution is to use federated learning or swarm learning to train neural networks. The goal of both approaches is to share only the learned weights of the neural network across different surgical centers while keeping each centre's data sets private. This creates a well generalised (global) model. To make this process even more secure, we want to incorporate the secure container environment from Scontain. Various validation techniques and metrics are known to evaluate the generated models. The goal is to compare the global model with different models in terms of metrics such as the F1 score (Dice Coefficient), the Intersection-Over-Union (IoU or Jaccard

Index), or pixel accuracy:

- Which are trained centralised on only one of the datasets.
- Which are trained centralised on all the datasets combined into one dataset (only possible when surgical centres are allowed to share data).
- Which are trained on a different dataset at the same domain.
- Which have different neural structure for the same task.

This experiment depends on access to, in some sense, Cloud computation resources, in that computation must be performed within dispersed computation centers of independent healthcare institutes, as well as a means to transmit data securely between them.

4.5.3 Involved Tools and Systems

In our experiments, the following Software tools and Systems will be used:

- Docker
- SCONE (for federated learning)
- Python
- Pravega (for distributed computing on data streams)
- GStreamer
- PyTorch (among other deep learning frameworks)
- ROS (Robot Operating System)

4.6 Testbed

We are going to provide a Cloud/Edge Testbed, required for the experiments and prototypes that will arise from the project. The Cloud testbed will deliver the essential components proposed by the reference architecture. This Testbed is going to validate and run the selected software platforms that will feed NEARDATA project.

KIO Networks, as a Service Provider, working on this type of initiative provides the company with growth both in knowledge and in integration with the rest of Service Providers that adopt technologies of this type, with the main objective of standardizing and industrializing services, thus building clouds integrated with load and data movement capacity between them. KIO Networks is very interested in the exploitation of Cloud technologies involving data management. As a company offering Cloud infrastructure to public institutions in Spain.

KIO Networks will provide Infrastructure Cloud Computing services on DataCenters like computing, storage and network resources. This service will be in multi-tenant mode, with virtualized environments with VMWare technology together with Kubernetes cluster services, high-performance storage (SSD).

The privacy and confidentiality of the data hosted in any cloud service are paramount. At KIO Networks we work using the strictest procedures to ensure that only the owner of the information has access to it. We periodically carry out audit processes that certify the validity of our procedures. Proof of this are our certifications in this area: ENS Categoría Alta (National Security Scheme, Highest Level). ISO 27001, ISO 27017, ISO 27018, ISO 22301, ISO 9001, PCI DSS.

4.6.1 KIO Networks Cloud Computing Architecture (VDC)

KIO Networks Spain Virtual Data Center (VDC) service will provide its consortium partners with the same infrastructure they use in their Data Center on a pay-per-use basis and as a Cloud Computing service. Virtual Data Center provides dedicated computing resources (vCPU, RAM), storage and connectivity, providing our partners with complete autonomy to generate the environments they need, being able to create, destroy or modify virtual servers and provide them with connectivity as needed at any time.

The platform is managed through a secure web portal that is intuitive to use and with a wide range of functionalities that allow system administrators to manage resources, servers and security policies from a single panel.

Kio Networks only work with leading manufacturers and providers in each segment, which is why we rely on top-level brands to support our platform. The VDC service is based on VCloud Director technology from VMWare depicted in Figure 41, the undisputed leading provider of virtualization solutions in the market. In addition to working with the most reliable hypervisor, the platform includes an API for integration with third-party deployment orchestration and automation systems.

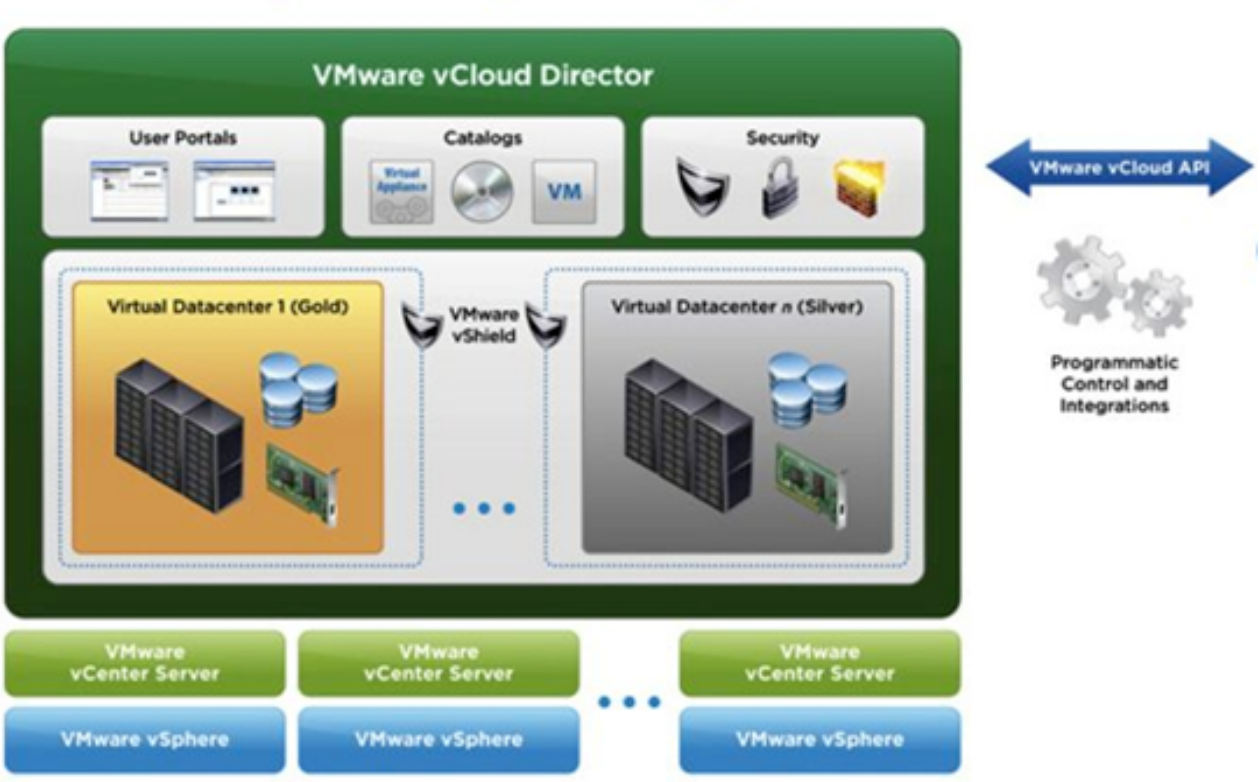


Figure 41: Highlevel Architecture of VCloud Director.

The VDC platform is delivered from Kio Networks TIER IV certified Data Centers in Murcia (Spain) or Querétaro (Mexico). Depending on the needs of each client, it is possible to hire VDC resources in Spain and/or Mexico, being able to connect both regions securely to allow communication between servers deployed in the two geographical areas. The TIER IV certification allows us to guarantee the highest levels of service availability: 99.95% uptime.

KIO Networks will provide an Integrated Networking and Perimeter Security Management. The VDC web control enables the administrator to manage virtual networks and perimeter security policies as shown in Figure 42. The platform delivers a VMWare NSX Edge Virtual Firewall by default, which includes the following features like NAT/PAT rules, L4 Firewall Rules, IPSec L2L tunnel man-

agement, DHCP server.

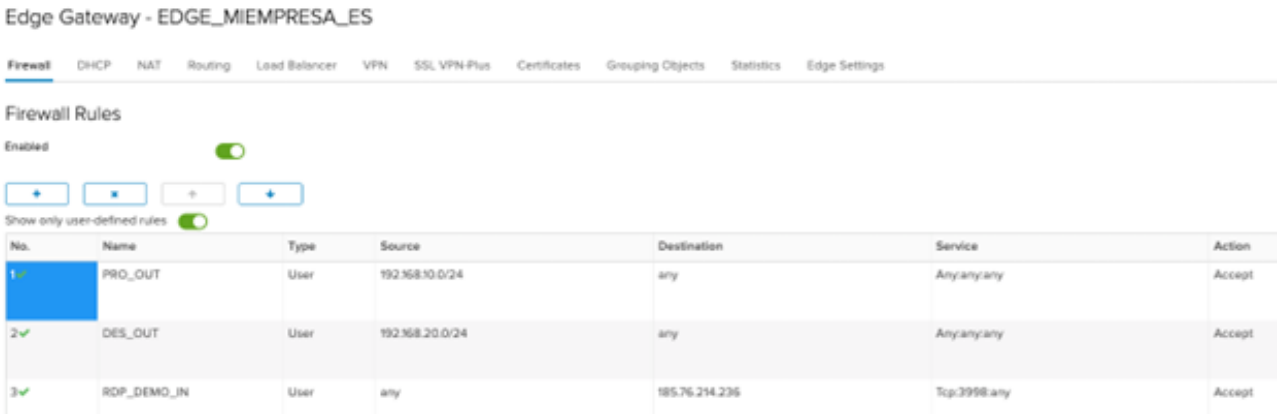


Figure 42: Edge capabilities.

The management of networks and Firewall policies complements the management of compute and storage resources, giving the VDC administrator the necessary autonomy to publish services, create DMZs where to isolate environments, etc. It is possible to complement or replace the services of the Firewall Edge with advanced Firewall/UTM solutions for those projects that require a greater level of depth in perimeter security. Consult with our Cloud experts the available options.

Support and service levels. The VDC service includes support and monitoring of the infrastructure 8x5. Kio Networks offers its customers 24-hour telephone support and access to a management panel for incidents and requests. The support team is made up of experts in the different areas that make up the VDC service. An escalation of up to 3 levels is offered depending on the complexity and implications of the incidents.

At Kio Networks, we work with business-critical IT environments, which is why we adapt our SLAs (Service Level Agreements) to the criticality and urgency of incidents. We offer response times according to the demands of our customers.

4.6.2 KIO Networks Cloud Computing GPU Architecture (VDC-GPU)

KIO Networks Cloud grows stronger together with NVIDIA, the most important international GPU provider at the moment. Thanks to this agreement, from KIO Networks we can offer advanced computing services based on GPU technology with the highest performance on the market. GPU-based computing technology enables utilization of the GPU's computing power for the most intensive work, while keeping CPU usage for operating system operations and desktop applications.

To understand the concept, we can say that a CPU has a few cores optimized for sequential serial processing, while a GPU has a huge parallel architecture consisting of thousands of smaller, more efficient cores. These cores are designed for parallelization or concurrency of tasks, thus the spectrum of work is reduced compared to the traditional CPU.

4.6.3 KIO Networks Cloud Computing S3 Architecture (S3)

NetApp ONTAP 9.8 software supports the Amazon Simple Storage Service (S3). ONTAP supports a subset of AWS S3 API actions and allows data to be represented as objects in ONTAP-based systems, including AFF, FAS, and ONTAP Select. NetApp StorageGRID® software is, and will remain, the NetApp flagship solution for object storage. ONTAP complements StorageGRID by providing an ingest and preprocessing point on the edge, expanding the data fabric powered by NetApp for object data, and increasing the value of the NetApp product portfolio. Figure 43 depicts the network architecture of KIO Networks Clients to have access to the S3 Service provided by NetApp ONTAP.

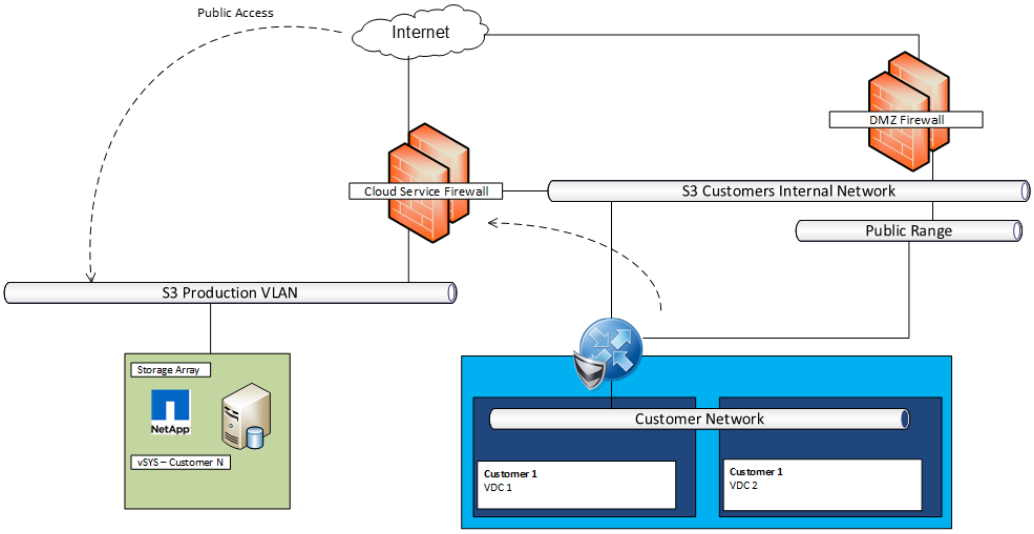


Figure 43: S3 Highlevel Architecture.

The S3 public service in KIO Networks will serve in two flavours:

- Hot Data (SANKLOUDNAS): This type of data will be fast access data on SSD disk for uses such as Financial Services, Online transactional processing (OLTP),High Performance Computing (HPC),Data lakes,Cloud native applications,Mobile applications.
- Cold Data (SANKLOUDBCK): This type of data will be backup data on a SATA disk to guarantee a coherent backup like Non-critical files or Sporadic files.

4.6.4 KIO Networks Cloud Computing Kubernetes Architecture

As part of the services offered by KIO Networks Spain within its datacenter is the container service based on Kubernetes technology. This service will be delivered to project partners who request it in the same way that KIO Networks Spain delivers it to the rest of its clients.

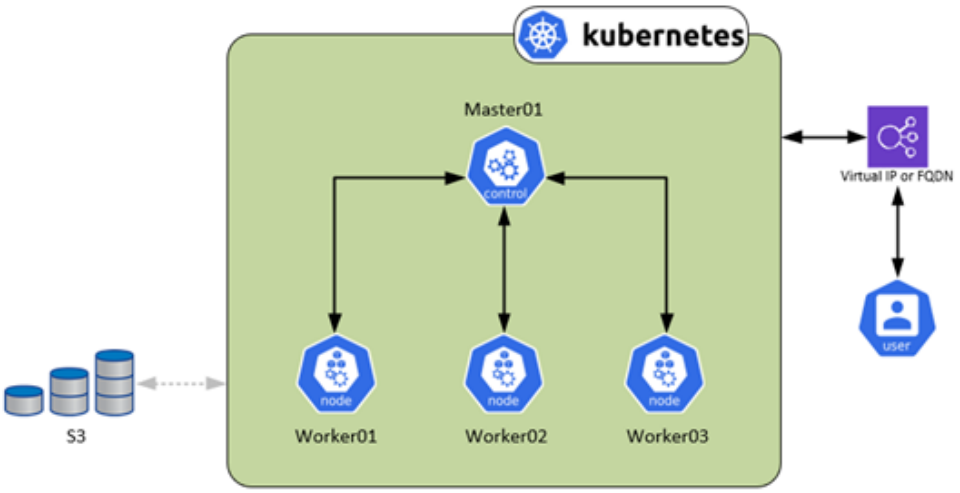


Figure 44: Kubernetes Highlevel Architecture.

The Kubernetes environment proposed for the project will be based on an architecture with the characteristics that can be seen in the Figure 44. It will be composed of:

- 1 Master server: According to the official Kubernetes documentation, the master server is a set of three daemons running on a single master node. These daemons are:
  - Kube-apiserver.
  - Kuber-controller-manager.
  - Kube-scheduler.
- 3 Worker-type servers: these are the nodes responsible for executing the applications in pods. These worker nodes are composed of two services:
  - Kubelet.
  - Kube-proxy.
- Possibility of connecting storage environments under the S3 protocol.
- Balancing service within the platform.
- Public Internet browsing IP.
- Internet flow according to the needs of the Partner.
- High performance storage based on SSD technology.

4.6.5 Partners requirements

KIO Networks have been held with each of the partners several meetings to determine the infrastructure requirements that KIO will need to provide to the consortium. The requirements were divided into Services, Computing, Storage and availability.

- Table 1 shows the technology used by each partner.

	Technology			
Partner	**VDC**	**VDC+GPU**	**S3**	**K8S**
PRAVEGA-DELL	X	X	X	X
SCONTAIN	X	X	X	X
LITHOPS	X	X	X	X
EMBL	X	X	X	X

Table 1: Technology requested per partner

- Table 2 shows the storage type used by each partner.

	Storage Type		
Partner	**SATA**	**SAS**	**SSD**
PRAVEGA-DELL	X	X	X
SCONTAIN	X	X	X (1TB)
LITHOPS	X	X	X (1TB)
EMBL	X	X	X (1TB)

Table 2: Storage type requested per partner

- Table 3 shows the computational resources used by each partner.

	Computational Resources	
Partner	**vCPUs**	**RAM (Gb)**
PRAVEGA-DELL	8 - 112	64 - 512
SCONTAIN	8 - 112	64 - 512
LITHOPS	300	64 - 512
EMBL	300	64

Table 3: Computational resources requested per partner

5 Confidential and ethical requirements for AI technologies

When designing AI technologies in healthcare, it is essential to address potential threat models that might be exploited by a malicious user, to ensure the security and privacy of patient data. Some of these threat models include:

- *Data breaches*: Data breaches occur when sensitive patient information is accessed, stolen, or compromised by an attacker. This can occur due to poor security measures or vulnerabilities in the system.
- *Adversarial attacks*: Adversarial attacks can also be used to manipulate the output of AI systems used in healthcare, potentially leading to incorrect or harmful diagnoses and treatments [40].
- *Model poisoning*: Model poisoning involves intentionally introducing malicious data into the training data used to train an AI model, with the goal of causing the model to produce incorrect or biased results [41].
- *Insider threats*: This refers to employees or contractors with authorized access to the AI system or patient data, who intentionally or unintentionally compromise the confidentiality or integrity of the data.
- *Legal and regulatory compliance*: AI systems used in healthcare must comply with legal and regulatory requirements, such as GDPR, to protect patient privacy and security [42].

Federated learning is a distributed machine learning methodology where data is distributed across data providers, where each participant holds custody of their data and its processing (modelling and prediction uses), avoiding data sharing or offloading [43]. Unlike centralised machine learning approaches, where all the data is transferred to a centralised system to be modelled, federated learning avoids data to leave the premises of each participant party. Each data provider models their data in their premises, and share across participants the modelling results or intermediate results, to be aggregated locally by each participant into a general model. Often, machine learning algorithms are coordinated step-by-step across participants, sharing intermediate results (models, weights, ...) to coordinate local models in the same training direction, but without sharing any training data. E.g., a consortium of hospitals might want to create a general model on certain data, but they can't share their patients data across partners. However, the hospitals can train local models without offloading any data outside their premises, sharing the resulting models across partners, who will aggregate their own models with the received ones. Or also, during local training, partners can coordinate to share intermediate algorithm values (e.g. weights or gradients on a neural network) at every step of the training algorithm, and aggregate them to have a global model at the end. Such distributed methods enforce model sharing against data sharing, where the only data transmitted across federated partners are the trained models or intermediate results during modelling algorithms. This implies that each federated partner must have capable computing infrastructures also enough storage for modelling and storing their data. Also, each partner must have a reliable connection in order to coordinate with other partners without becoming a bottleneck for the process.

As federated learning avoids the distribution of data, aligned with the principle of data minimization, the risk of a privacy breach is reduced. However, shared models can also be susceptible to privacy attacks, such as *membership inference attacks* to retrieve sensitive data. The attacker can discover or reconstruct the sensitive data used to train the models from the knowledge of the algorithm and architecture used without the need to know the inner parameters. For such cases, there's the option of removing singular cases from the training dataset, avoiding federated partners to discover such peculiarities by interpreting the model. Another option focuses on introducing white noise or neutral synthetic data for training, masking the real data. In this way we can achieve the *differential privacy*, from these mechanisms we can ensure the privacy of individuals in the datasets. To these measures we can also implement standard security measures for transmission, such as encryption, access control and differential privacy methods. It is essential that in order to guarantee the privacy

and confidentiality of the data, the models do not reach the point of overfitting. Ensuring a fully generalized model on the training data will largely prevent inference attacks from succeeding.

Previously introduced security attacks such as model poisoning can affect the initial stages of Federated Learning. One of the most commonly used types of data poisoning attacks is label-flipping attacks where the adversary simply modifies the training labels. Thus, the model is trained on a poisoned dataset without knowing which labels have been modified. When we talk about Federated Learning we must take into account that due to its decentralized and distributed nature it is more susceptible to any security attack than in centralized architectures. To try to mitigate these attacks it is necessary that the servers can see or identify the individual updates of each client (even if they are distorted with noise). For this purpose, there are different methods or mechanisms that allow filtering or detecting corrupted updates that correspond to security attacks such as trimmed-mean or Fool's Gold, which are based on establishing zero trust in updates that servers may believe to be of value but are actually malicious updates, which are easy to implement and very effective.

Federated learning addresses threat models that can increase ethical concerns on individuals privacy (e.g., patients, citizens, clients, etc.), while allowing organizations to train a consensus model, much more proficient at solving a task at hand than isolated models trained within each organization. Through a federated architecture, organizations keep control of their data by preventing any transmission. In the scenario of medical data, data providers (hospitals) retain ownership and control of their medical data, reducing risks related to third-party data breaches and insider threats. Furthermore, lack of data transfers also alleviates compliance issues related to regulatory and legal requirements for patient privacy and security. In addition, federated learning can even provide more robust and resilient models by minimizing local biases and noise on data, also allowing models to continue functioning even if one of the partners is disconnected from the system or becomes compromised. In the context of the aforementioned threat models, this means that it also reduces the influence of data manipulation and model poisoning - the larger the federation, the more robust the consensus model.

As follows in this section, we are detailing the specific ethical requirements for the provided use cases.

5.1 Clinical Sequencing of Human Pathogens Use Case

In principle, this use case would seem relevant to the confidentiality and ethical issues just highlighted — and also prone to a number of additional concerns. For instance, the potential presence in the sample of sequencing reads originating from the host might lead to the patient being identified; that's because, like fingerprints, the pattern of genomic variants can be considered as characteristic of each individual, and can be easily used to look up the patient's identity in genealogy or other databases. Another risk, seen that sample analysis implies a large number of potential actors (GPs, clinicians, hospitals, analysis labs, epidemiologists, national- and country-level public bodies) in a federated environment, is that data and metadata might be connected in order to identify the sample's owner outside of the approved data management structure. As a matter of fact, separation of metadata from the sample by barcoding, and anonymisation of sequencing data by discarding reads that originate from the host, are performed right at the beginning of the process, in order to avoid concerns when sample-related data circulates through the analysis workflow and gets accessed by different actors in the federated environment. This design allows UKHSA to make sequencing reads (the ones originating from the pathogen, from which host reads have been purged) publicly available on repositories such as NCBI or ENA, while metadata is kept secret due to it being ripe with personally identifiably information which the patient has in most cases not agreed to share for research or other purposes.

In practice, it would be extremely difficult for UKHSA to make available any of the metadata to this consortium. In order to solve this problem, it has been decided that all the experiments performed for this project will be conducted on synthetic metadata. The artificial generation of non-personal metadata (such as population geography, demographics, epidemiology, calendar and an-

nual periodicity) will be realistic, in the sense that all relevant metadata fields will be generated from realistic distributions measured from the UKHSA's internal metadata database. In order for these data to maintain the privacy preservation, they will be distorted by adding noise before generating the synthesized data, thus preventing the synthesized generation from generating original data in an unambiguous way. This should allay fears of inadvertent leaks of personally identifiable information from the UK to the EU for the duration of this project.

5.2 Variants Interaction Use Case

The high level of sensitivity of biomedical data converts the access, management and use of this information into one of the main concerns of data providers, thus demanding understanding of the responsibilities derived from data accession and the enforcement of appropriate security practices. For this reason, for the variant interactions use case, diverse agreements have been legally signed between the Barcelona Supercomputing Center (BSC) and the different data providers, including the National Institute of Health and the Sanger Institute. As a result of these agreements:

- To ensure the **confidentiality of the data**, only some of the participants from the BSC have been granted controlled-access to the genotypic and phenotypic information of the individual cohorts that are going to be analysed, exclusively, to explore the effect of variant interactions (FUSION, GENEVA, NUgene, GERA and WTCCC). Additionally, these BSC participants have compromised to avoid any attempt of identification, and even the generation of any type of information that can facilitate the identification of any of the patients included in these studies. We also envision the use of confidential compute which provides process isolation and hence adds another layer of security to ensure confidentiality of data. This will be applied to **data at rest**, i.e. stored at stable/persistence storage by using transparent file encryption, for **data in transit**, i.e., transferred through network connection as well as **date in use**, i.e., that is currently be processed by the machine learning/AI frameworks.

- To ensure the **security and privacy of the data** the BSC has settled a security plan, which involves the procurement of diverse tools and infrastructure to guarantee the compliance of privacy legislation, secure transfer, retention and disposal of the data during the project. This security plan is aligned with the diverse security principles and recommendations for data, access, and physical security required by the different data providers.

- To ensure **data and model integrity** against data manipulation and model poisoning, the machine learning methods implemented in this use case will follow the current standard recommendations and practical suggestions discussed above to prevent security attacks. Thus, not only guaranteeing the transparency of the models but also avoiding any possible ethical issues derived from the analyses regarding different socio-technical scenarios.

Therefore, the genomics data from Variant Analysis will be subject to:

- *Patient confidentiality* through the existing embargo of the complete datasets. The data is retrieved from a curated genomics repository, where only personnel screened by an admission committee backed by an ethics committee has granted access. And for the current use case, the personal information from the patients is NEVER retrieved, keeping only genomics data for experimentation in the protected environment from the Barcelona Supercomputing Center. None of these genomics data will ever leave BSC premises.

- *Data protection* through the controlled access to the BSC systems and data repositories. No sensitive data or data derived from sensitive data will ever leave BSC premises.

- *Model anonymity* by checking that the shared machine learning models are protected against membership inference attacks with the introduction of synthesized data or white noise. Seeking fully generalized models to avoid model spicification that would facilitate reconstruction of sensitive data.

Finally, the use case will generate derived data of two kinds: potential sensitive and telemetry data. Potential sensitive data is the intermediate data generated from processing genomics data, and will never leave the processing premises. Telemetry data is data from the systems processing data, totally agnostic of the potential sensitive data and genomics data, and will never contain sensitive data. Examples of such telemetry data there is "Amount of CPU/Memory used to process a patient", "Time to process a single patient data", "Accuracy of a model examining a certain variant".

5.3 Transcriptomics Atlas Use Case

Next-generation sequencing produces large amounts of data, which increases complexity of data storage, processing and analysis. To add to this complexity, there are many security risks related to sequencing datasets, that can influence both research participants and patients. Such data can be subject to many adversary behaviours with different goals, including [44]:

- *Identity tracing*: obtaining anonymized sequencing data and re-identifying the person the data belongs to, based on e.g. demographic data.
- *Attribute disclosure attack*: the adversary tries to associate sequencing data with sensitive attributes of the victim, such as phenotype, drug abuse or a specific disease.
- *Completion attacks*: reconstructing complete sequencing data out of partial information available on different sequencing projects or based on a family member sequencing data.

These goals can be accomplished by analysing metadata associated with genomic samples, the samples itself, or by combining samples with publicly available information from other repositories. There are also attacks focused on deriving patient-specific information from aggregate statistics.

Another important factor when handling transcriptomics data, especially in cloud-environment is legal regulations related to data sharing. The GDPR regulations [42] allow members to impose additional constraints on data sharing in their states. In practice this means, that in some countries, the data can be shared quite freely, but in others there are certain restrictions. For example in Netherlands, institutions can access the data if they use this data only for statistical or scientific purposes, whereas in Germany it must remain impossible to reidentify the involved patients [45].

Federated learning workflow is a part of experiments conducted in the Transcriptomics Atlas use case and can efficiently alleviate problems with both factors. Risks related to the identity tracing, attribute disclosure and completion attacks can be mitigated by preventing any data samples from leaving the organisations they were collected in the first place. With no data transfer, there is also no risk of breaching GDPR constraints on the exchange of medical data, which simplifies the implementation of transcriptomic data analysis systems in a real-world setting.

In the use case, we plan 2 experiments: (1) building transcriptomics atlas from data publicly available in NCBI SRA and (2) FL experiment using synthetic data, in both of them no use of sensitive data is planned. The goals of these experiments will be to evaluate the FL technologies and their possible extensions with trusted execution environments with respect to performance, ML model quality and potential risks. The results of these experiments carried out in the project with the use of non-sensitive data will provide insights into potential threats and ethical issues when putting these solutions into production in the future.

5.4 Metabolomics Use Case

For the metabolomics use-case we foresee no ethical issues because for the project purposes we will use only public data from METASPACE which is already shared by the data owner (METASPACE users) under the CC BY 4.0 license [13].

[13] https://metaspace2020.eu/terms

5.5 Confidentiality and ethical considerations in Surgery Use Case

Surgical data-driven machine-learning (ML) methods for computer-guided (robotic) surgery (CGS) rely on large amounts of multi-centric surgical data with high variance to train the algorithm and improve its accuracy. In this project, the surgery use case operates mainly on video data, which raises additional confidentiality and ethical considerations beyond those that apply to CGS in general. Some of the key requirements for the ethical use of surgical data-driven deep learning models in CGS include:

- *Patient and surgical staff confidentiality:* It is critical to protect patient and surgical staff confidentiality when using surgical data, especially video data, to train ML models. This includes ensuring that all surgical data used in the training process is de-identified, i.e. removing all frames from outside the abdominal cavity and all meta-data related to the patient, surgical staff and time/place of operation, and securely stored to prevent unauthorized access.
- *Informed consent:* Patients must be fully informed about the use of their data in the development of ML models for CGS. This includes explaining the purpose of the research, the potential benefits and risks, and the safeguards that will be put in place to protect their privacy.
- *Data protection:* All data used to train ML models must be protected and secured to prevent unauthorized access or use. This includes implementing appropriate access controls, using encryption, and ensuring that data is stored and used on secure servers.
- *Algorithm transparency:* The ML algorithm used in CGS must be transparent and explainable to ensure that surgeons can understand how the technology is making decisions. This is essential for maintaining trust and ensuring that the technology is used ethically and appropriately.
- *Quality assurance:* A quality assurance program must be in place to ensure that the ML model is being used appropriately and that patient outcomes are being monitored. This includes regular audits of the technology and its use, as well as ongoing training and education for surgeons and other healthcare professionals.
- *Risk management:* To mitigate the effect of data leaks in the course of this project, we will focus on using (a) publicly available datasets, (b) data collected from phantom trials (e.g. silicon phantoms), (c) synthetically generated datasets and (d) utlize only de-identified videos, were all frames from outside the abdominal cavity and all meta-data related to the patient, surgical staff and time/place of operation have been removed.

To ensure the confidentiality and ethics of data-driven ML methods in CGS, the legal, data protection, and research staff at the NCT/DKFZ work closely with the surgeons at the multisite centers throughout the project pipeline on the developed communication platform from NCT/DKFZ. The ethical application and patient information/consent should be reviewed by the ethics committee prior to surgical data collection. In addition, the Data Transfer Agreement and the Joint Controller Agreement must be signed by all relevant partners. On the technological side, the surgical use case clearly requires confidential compute (WP4) for sensitive information, high-performance stream analytics close to the data. Furthermore, federated learning is required to facilitate surgical data sharing between different healthcare centers. To mitigate the threats to the learning models that can compromise users, it is crucial for us to implement robust security measurements, including encryption and authentication protocols, to protect surgical data privacy and ensure secure communication channels. Regular auditing and validation of the federated learning process should be performed to detect and mitigate potential attacks or model vulnerabilities.

As previously mentioned, we envision the use confidential compute as a technological asset to address some of the confidentiality and integrity issues raised for the different use case above. Confidential compute such as carried out through Trusted Execution Environments (TEEs) enables users

to completely isolate applications on process levels such that even privileged users such as administrators are neither able nor privileged to access or manipulate the program code and data on the physical node.

5.6 Summary and Contingency of Ethical Issues

Table 4 summarizes the ethics concerns on data and technologies used to process such. There, the different use cases define the data risks, retrieval or acquisition conditions related to such data, contingency measures taken towards potential risks, the agreed sharing policy between partners and third parties, and the involved technologies dealing with the data.

Use Case	Data Risks	Data Conditions	Contingency Measures	Sharing Policies	Involved Technologies
Clinical Sequencing of Pathogens (UKHS)	Identifying a patient from variants fingerprint present in genomics data	Patients provide consent permissions to UKHSA to process their samples	Sequencing reads originated from the host are removed and discarded at the beginning of the analysis pipeline	No data is shared	Custom scripts (UKHS)
	Correlating the results of the test with the patient's personal information	Patients grant UKHSA permission to process their samples and use their personal information in relation to their clinical history	Sample metadata and personal information are decoupled at the beginning of each analysis pipeline	No data is shared	Custom scripts (UKHS)
	Sharing of personally identifiable information/metadata from the UK to the EU	Patients grant UKHSA permission to use their personal information in relation to their clinical history	No actual data will be used or shared during the project. All metadata used during the project will be simulated	No actual data will be used or shared during the project. All metadata used during the project will be simulated	Custom scripts (UKHS)
Variant Interactions Analysis (BSC)	Datasets include patients genotype and phenotype (identification)	Security plan in place, only specific people have been granted access to full data	Patients identification is removed from the processing pipeline	Only synthetic generated data is shared	Apache Spark, Lithops, PySpark, HDFS.
Surgery Use Case (NCT)	Video frames could identify patient/-surgical staff	Critical data will not leave NCT and will not be shared with unauthorized people	Critical frames will be removed from videos	Share public/synthetic/phantom data only with partners	Pravega, GStreamer, Python, Pytorch, and Flower
Metabolomics (EMBL)	No risks on data	Public data from METASPACE. Provided under CC BY 4.0	Does Not Apply	Data is shared under Creative Commons "BY" 4.0	METASPACE, Lithops
Transcriptomics Atlas (SANO)	No risks on data	Data obtained from public repositories or simulated data	Does Not Apply	Does Not Apply	Python, Pytorch, Flower, SCONE, Amazon Web Services, SRA-Toolkit, Salmon, STAR, Docker, Lithops

Table 4: Summary of Use cases versus Risks, Conditions, Contingencies and Technologies

As indicated on Table 4, the main potential risk of data identification is covered by the policies of data acquisition, where only authorised personnel with explicitly authorised purposes can access this data; sharing and distributing this data outside the technical facilities is totally restricted, with security plans to prevent intrusions and leakage; any data publication for dissemination and

demonstration purposes will be done using exclusively synthetic data; and all technologies are self-contained in the data tenant premises, where no piece of software will connect to third-party systems. Furthermore, NEARDATA partners have a specialised team on data security and control, committed to prevent any potential issue beforehand. **Any potential risk is covered** by the local (partner) and global (consortium) contingency plans (data management plan and security plan).

Finally, regarding to the use of AI, all methods and algorithms are designed and purposed for generating data of clinical value for medical professionals in patient evaluation and treatment prescription. No AI or algorithm is designed to target patients nor profile them for purposes other than the agreed by the patients and ethical committees from the data providers.

6 Conclusions

The deliverable presents the initial specifications of the global NEARDATA architecture as well as the different components that form it and all the use cases that are part of this project.

The requirements and specifications described in the document are intended to ensure compliance with all the main objectives presented in the project. In this way, it is intended to ensure that our novel NEARDATA platform meets the expectations in terms of designing an extreme near-data processing platform that enables distributed and federated data consumption, mining and processing without the need to master the logistics of accessing data across heterogeneous locations and datasets.

We clearly specified the following points in this deliverable:

- A new architecture has been presented based on the theoretical proposal with the incorporation of the components presented in NEARDATA. In addition, different life cycles of the architecture have been shown according to the type of execution (Basic or Confidential & Federated computing).
- The components that form the global NEARDATA architecture have been detailed as well as the software frameworks for its implementation. Then, the integration of these with the different use cases have been described.
- For each use case the requirements, specifications and objectives have been collected and the characteristics of the datasets, the experiments and their pipelines have been presented in an extended way.
- For the evaluation of each use case, different test environments and experiments to be performed have been presented respectively. Also, our partner KIO has provided different environments according to the requirements of each partner.
- For the sensitive health data, ethical and confidentiality requirements have been exposed to guarantee the correct use, protection and privacy of the data.

A Appendix: NEARDATA APIs

A.1 Data Plane APIs

A.1.1 Data Catalog APIs

METASPACE API. METASPACE has an API which provides full access to the data and processing capabilities. The API is documented at `https://metaspace2020.readthedocs.io` with the examples including how to fetch dataset annotations, how to fetch dataset metadata, and how to submit a dataset. The index provides a comprehensive list and descriptions `https://metaspace2020.readthedocs.io/en/latest/genindex.html`. A few illustrative examples are included in the table below:

API	API Call	Description
METASPACE API	`submit_dataset`	Submit a dataset for processing in METASPACE
	`update_dataset`	Updates a dataset metadata and/or processing settings. Only specify the fields that should change. All arguments should be specified as keyword arguments, e.g. to update a dataset adducts
	`annotations`	Fetch dataset annotations
	`isotope_images`	Retrieve ion images for a specific molecular formula and adduct
	`GraphQLClient`	Client for low-level access to the METASPACE API, for advanced operations such as querying and search

Table 5: Examples of the calls from the METASPACE API.

A.1.2 Data Connectors APIs

Lithops. The role of Lithops within NEARDATA is defined as a serverless data processing platform. Lithops allows the user to run local python code massively on different cloud platforms. As mentioned in section 2.1, the core object *Function Executor* is in charge of this task. This object allows to perform calls to the Lithops API to run parallel tasks.

Lithops is shipped with two different high-level Compute APIs, and two high-level Storage APIs. Within the context of NEARDATA we will benefit from one of each type.

Compute API. Allows the user to invoke parallel tasks on the selected computational backend. The following is a description of the Futures API, which is based on the Python `concurrent.futures` library:

API	API Call	Description
Futures API	`call_async(func, data)`	Method used to spawn one function activation
	`map(func, iterdata)`	Method used to spawn multiple function activations
	`map_reduce(map_func, map_iterdata, red_func)`	Method used to spawn multiple function activations with one (or multiple) reducers
	`wait()`	Wait for the functions activations to complete. It blocks the local execution until all the function activations finished their execution
	`get_result()`	Method used to retrieve the results of all function activations. The results are returned within an ordered list, where each element of the list is the result of one activation
	`plot()`	Method used to create execution plots
	`job_summary()`	Method used to create a summary file of the executed jobs. It includes times and money
	`clean()`	Method used to clean the temporary data generated by Lithops

Table 6: Lithops: Futures API description.

Storage API. Users can perform calls using this API to operate with the storage backend selected. A detailed view of the Storage API can be found in the following table:

API	API Call	Description
Storage API	`put_object(bucket, key, data)`	Method used to add an object to a bucket of the storage backend
	`get_object(bucket, key)`	Method used to retrieve objects from the storage backend
	`delete_object(bucket, key)`	Method used to remove objects from the storage backend
	`delete_objects(bucket, key_list)`	Method used to delete multiple objects from a bucket using a single HTTP request
	`head_object(bucket, key)`	Method used to retrieve metadata from an object without returning the object itself
	`head_bucket(bucket)`	Method used to determinate if a bucket exists and the user has permissions to access it
	`list_objects(bucket, prefix)`	Method used to obtain a list with the name (key) and size of all the objects in a bucket
	`list_keys(bucket, prefix)`	Method used to obtain a list with the names (keys) of all the objects in a bucket

Table 7: Lithops: Storage API description.

Serverless Data Connector. The Serverless Data Connector's role in NEARDATA is defined as serverless data connector between object storage and the different Data Processing Platforms used in the use cases. Serverless Data Connector will allow efficient access to unstructured data thanks to the followings APIs.

API	API Call	Description
Partitioning API	`create_object(data_format, file_key)`	Method to create a Generic Partitioning Object
	`extract_metadata(processing_platform)`	Method used to extract the metadata from the original file
	`get_metadata(metadata_key)`	Method used to get the metadata extracted from the original file
	`generate_partitions(method, metadata_key)`	Method used to create partitions by byte ranges from the extracted metadata

Table 8: Serverless Data Connector: Generic Partitioning Object API description.

In the following table we can find the data connectors' API suggested for the use cases:

API	API Call	Description
Storage API	`ingest_data()`	Method use to ingest data in a parallelized
	`clean_data(strategy, data)`	Method used to remove or modify data coming from incomplete data of a patient. Different strategies will be implemented and will be of choice
Data API	`extract_data(features, filters)`	Method to extract relevant data based on the filters and features of interest.
	`data_transform(dataset, format)`	Method to transform the way data is presented to the use cases. This method serves to be able to store data in a single way for each use cases, and allow each of them to read the data in different formats.
	`merge_data(dataset, strategy)`	Method to merge the different splits of data processed in parallel into a single data. Different strategies will be devised.
	`interpret_data(data)`	Method to interpret the data according to each use case needs.

Table 9: API description for use-cases data connectors.

A.1.3 Streaming APIs

The streaming storage fabric in NEARDATA is provided with Pravega. At its core, Pravega is a pure tiered storage system for data streams that seamlessly and transparently moves data to a long-term storage tier, such object store. As we describe next, on top of the stream abstraction, Pravega provides a wide variety of APIs that can meet the requirements of multiple streaming applications.

Stream reader and writer APIs. One of the key design features of Pravega is that it unifies data access to real-time (or "recent") and batch (or "historical") data. This has an immediate impact on the API that clients applications use, as the same read call will enact Pravega fetching data recently cached or data stored in long-term storage, while the user is oblivious to the location of data. As visible in Table 10, Pravega provides async APIs for writing and reading data from a stream. Async APIs provide applications with the possibility of running non-blocking code to perform stream IO, which is key for high concurrency and parallelism. In addition to writing and reading events directly to/from a stream, Pravega also provides the concept of "transaction". The idea is to write a group of events as a whole on a stream atomically. This is necessary for providing exactly-once semantics on stream processing pipelines, in which the sink needs to be coordinated with the analytics engine to avoid missing events or writing them twice downstream.

API	API Call	Description
EventStreamWriter<T>	`CompletableFuture<Void> writeEvent(String routingKey, T event)`	Write an event to the stream.
	`CompletableFuture<Void> writeEvents(String routingKey, List<T> events)`	Write an ordered list of events to the stream atomically for a given routing key.
EventStreamReader<T>	`EventRead<T> readNextEvent(long timeoutMillis) throws ReinitializationRequiredException, TruncatedDataException`	Gets the next event in the stream. If there are no events currently available this will block up for `timeoutMillis` waiting for them to arrive. If none do, an `EventRead` will be returned with `null` for `EventRead.getEvent()`. (As well as for most other fields) An `EventRead` with `null` for EventRead.getEvent() is returned when the `Reader` has read all events up to the configured end `StreamCut` specified using `ReaderGroupConfig`. Note: An EventRead with `null` for `EventRead.getEvent()` is returned when `EventRead.isCheckpoint()` is true.
	`T fetchEvent(EventPointer pointer) throws NoSuchEventException`	Re-read an event that was previously read, by passing the pointer returned from `EventRead.getEventPointer()`. This does not affect the current position of the reader. This is a blocking call. Passing invalid offsets has undefined behavior.
TransactionalEventStreamWriter<T>	`Transaction<T> beginTxn()`	Start a new transaction on this stream. This allows events written to the transaction be written an committed atomically. Note that transactions can only be open for `EventWriterConfig.transactionTimeoutTime`.
	`Transaction<Type> getTxn(UUID transactionId)`	Returns a previously created transaction.
Transaction<T>	`void commit() throws TxnFailedException`	Causes all messages previously written to the transaction to go into the stream contiguously. This operation will either fully succeed making all events consumable or fully fail such that none of them are. There may be some time delay before readers see the events after this call has returned.
	`void abort()`	Drops the transaction, causing all events written to it to be deleted.

Table 10: Pravega main stream reader and writer APIs.

Byte reader and writer APIs. In many use cases, stream events are small pieces of data with a particular format (e.g., values representing metrics, small text messages or log lines, json objects, etc.). However, it is also possible that analytics applications need to write some large data object. To avoid having to use an external file or object storage service, Pravega provides the "byte API", which allow applications to write arbitrarily large data objects via Pravega (see Table 11). Behind the scenes, Pravega writes such data objects on top of stream segments. This is a low level API intended to be exploited by higher-level writers and readers. For instance, a good example of exploiting the Pravega byte API is the GStreamer[14] connector for Pravega, whch will be used for the computer-assisted surgery video analytics. This connector provides video frame formatting that writes data via the byte API.

[14] `https://github.com/pravega/gstreamer-pravega`

API	API Call	Description
ByteStreamWriter	`public abstract void write(byte[] b, int off, int len) throws IOException`	Writes the provided data to the segment. The data is buffered internally to avoid blocking. As such it cannot be assumed to be durably stored until a flush completes. It is intended that this method not block, but it may in the event that the server becomes disconnected for sufficiently long or is sufficiently slow that that backlog of data to be written becomes a memory issue.
	`public abstract void flush() throws IOException`	Blocks until all data written has been durably persisted.
ByteStreamReader	`public abstract int read(byte[] b, int off, int len) throws IOException`	If `available()` is non-zero, this method will read bytes from an in-memory buffer into the provided array. If `available()` is zero will wait for additional data to arrive and then fill the provided array. This method will only block if `available()` is 0. In which case it will block until some data arrives and return that (which may or may not fill the provided buffer).
	`public abstract void seekToOffset(long offset)`	Seeks to the provided offset (It can be anywhere in the segment). Future read calls will read from this offset.

Table 11: Pravega byte stream reader and writer APIs.

Synchronization APIs. Keeping a consistent shared state across multiple processes is also a common needs in many analytics workloads, such as serverless functions that do not have native means to share state consistently. Pravega provides an API superficially for that: the "state synchronizer". This API allows a group of processes to share some state and apply updates in a consistent fashion via optimistic concurrency. The state synchronizer API also builds on top of Pravega streams and makes uses of conditional appends to linearize writes from multiple processes to the shared state.

API	API Call	Description
StateSynchronizer<T>	`void initialize(InitialUpdate<T> initial)`	This method can be used to provide an initial value for a new stream if the stream has not been previously initialized. If the stream was already initialized nothing will be changed, and the local state will be updated as though `fetchUpdates()`.
	`void fetchUpdates()`	Fetch and apply all updates needed to the state object held locally up to date.
	`void fetchUpdates()`	Fetch and apply all updates needed to the state object held locally up to date.
	`T getState()`	Gets the state object currently held in memory. This is a non-blocking call.
	`void updateState(StateSynchronizer.UpdateGenerator<T> updateGenerator)`	Creates a new update for the latest state object and applies it atomically. The `UpdateGenerator` provided will be passed the latest state object and a list which it can populate with any updates that need to be applied. These updates are recorded and applied conditionally on the state object that was passed to the function being up to date. If another process was applying an update in parallel, the state is updated and `updateGenerator` will be called again with the new state object so that it may generate new updates (which may be different from the one it previously generated). By re-creating the updates in this way, consistency is guaranteed. When this function returns the generated updates will have been applied to the local state.
	`void updateStateUnconditionally(Update<T> update)`	Persists the provided update. To ensure consistent ordering of updates across hosts the update is not applied locally until `fetchUpdates()` is called.

Table 12: Pravega `StateSyncrhonizer` APIs.

Key/Value Table APIs. Another common abstraction required by many types of workloads and applications is the Key/Value abstraction. Similar to having a durable hash table, Pravega offers a Key/Value API for storing and retrieving data associated with a key. Keys and values are also stored in data streams behind the scenes. This Key/Value API may help analytics pipelines to avoid having to set up an additional K/V store service. The API allows key updates to be conditional or unconditional.

API	API Call	Description
KeyValueTable<T>	`CompletableFuture<Version> update(TableModification update)`	Performs a specific TableModification to the KeyValueTable, as described below: i) If update is a Insert, this will be interpreted as a Conditional Insert, so the `TableEntryUpdate.getValue()` will only be inserted (and associated with `TableModification.getKey()` if there doesn't already exist a `TableKey` in the `KeyValueTable` that matches `TableModification.getKey()`. ii) If update is a `Put`, this will be an update. If `Put.getVersion()` is null, this will be interpreted as an Unconditional Update, and the `TableEntryUpdate.getValue()` will be associated with `TableModification.getKey()` in the `KeyValueTable` regardless of whether the `Key` existed before or not. iii) If `Put.getVersion()` is non-null, this will be interpreted as a Conditional Update, and the `TableEntryUpdate.getValue()` will only be associated with `TableModification.getKey()` if there exists a `TableKey` in the `KeyValueTable` whose `Version` matches `Put.getVersion()`. iv) If update is a `Remove`, this will remove the `TableModification.getKey()` from the `KeyValueTable`. If `Remove.getVersion()` is null, this will be interpreted as an Unconditional Removal, so the `Key` will be removed regardless of its `Version`. If `Remove.getVersion()` is non-null, this will be interpreted as a Conditional Removal, and the `Key` will only be removed if it exists and its Version matches `Remove.getVersion()`.
	`CompletableFuture<TableEntry> get(TableKey key)`	Gets the latest value for a `TableKey`.

Table 13: Pravega main Key/Value Table APIs.

A.2 Control Plane APIs

In the following table we show the APIs to provide the AI-based optimizations for Cloud/Edge workflows.

API	API Call	Description
ML API	`train_model(dataset, model, input_parameters)`	Outputs the trained model using the model and dataset provided
	`infer_model( input_parameters, trained_model)`	Computes the inference for the input parameters provided using the previously trained model. The trained model should be obtained with the train_model call.
	`validate_model( validation_dataset, trained_model)`	Validates the trained model using the validation dataset provided
Resource Management API	`schedule_workload( workload, waiting_queue, running_queue, policy))`	Schedules the workload based on the given policy and the current waiting and running queues of workloads.
	`allocate_resources( workload, policy, SLA)`	Allocates resources for the provided workload using a specific policy and the SLA requirements.

Table 14: AI-Based optimizations APIs.

A.2.1 SCONE CAS API

CAS is one of the central components within NEARDATA as it defines the security policies which ensures that files read and written by the various tools within NEARDATA such as Pravega, Spark etc. are encrypted properely. It furthermore ensures that processes run in Trusted Execution Environments such as Intel SGX and provides access control for network connections. CAS provides two different interfaces to configure applications. Either via CLI interface and so called Policy session files written in YAML format or via the REST interface of CAS.

CAS itself is protected by running inside of a production SGX enclave and encrypting its database which is stored in the filesystem. Encrypting files is transparent to applications running in the context of a CAS: CAS attests the application and then passes its filesystem keys to the application. Actually, it passes the keys to the SCONE runtime of the application and the runtime transparently encrypts files on writes and decrypts files on reads. CAS itself can and should be attested to ensure that it can be trusted.

API	API Call	Description
CAS CLI	`session create -use-env session.yaml`	Creates and uploads the provided session.yaml file with environment variables replaced
	`scone session update -use-env session-update.yaml`	Updates an existing SCONE session based on the policy defined in the session-update.yaml.
	`scone session delete <SESSIONID>`	Removes a previously posted/used session
CAS REST API	`POST /session`	Creates a session based on the provided payload of the POST request
	`PUT /session`	Updates an existing SCONE session based on the policy defined in the payload of the PUT request

Table 15: CAS APIs.

A.2.2 Keycloak API

Keycloak is an additional mechanism to enforce an authentic identification of people operating the systems within NEARDATA and their corresponding access level to other systems and resources protected within the perimeter. Keycloak will issue access tokens with the specific roles that a system administrator has attributed to the user requesting it. The other system that is receiving this token will obtain a validation token from Keycloak, that will attest its authenticity and that the roles therein are correct. This provides protection to an application and to its resources being accessed by only the authorized people.

Keycloak runs protected by mechanisms of confidential computing, i.e. protected by an enclave in conjunction with SGX (the TEE hardware enabling confidentiality), and is also additionally protected by CAS, hence all the configuration, credentials and keys are provisioned via attestation.

Both access token and validation token are issued via REST communication, using secure channel TLS enabling HTTPS server. Here are the access token API and validation token API. We have plans to adopt it as an adjunct system for the Confidential Data Orchestration.

API	API Call	Description
Access Token	Method: **POST**; URL: `https://KEYCLOAK.SERVER/realms/neardata/protocol/openid-connect/token`; Header: `Content-Type = application/x-www-form-urlencoded`; Data set: `client_id = $APPLICATION PROTECTED$, username = $USER LOGIN NAME$, password = $PASSWORD$, grant_type = password`	Obtains one access token with the access level defined to that user
Validation Token	Method: **POST**; URL: `https://KEYCLOAK.SERVER/realms/neardata/protocol/openid-connect/userinfo`; Header: `Authorization = Bearer $ACCESS TOKEN$`	Obtains one validation token corresponding to the access token sent

Table 16: Access Token and Validation Token acquisition APIs

References

[1] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in 2015 IEEE Trustcom/BigDataSE/ISPA, vol. 1, pp. 57–64, 2015.

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

[3] "Lithops." https://lithops-cloud.github.io/, 2021.

[4] "concurrent.futures library." https://docs.python.org/3/library/concurrent.futures.html, 2023.

[5] "multiprocessing library." https://docs.python.org/3/library/multiprocessing.html, 2023.

[6] "boto3 library." https://boto3.amazonaws.com/v1/documentation/api/latest/index.html, 2023.

[7] "Amazon web services. registry of open data on aws." https://registry.opendata.aws/, 2022.

[8] "Pravega - pdp 48 (key value tables beta 2)." https://github.com/pravega/pravega/wiki/PDP-48-(Key-Value-Tables-Beta-2), 2023.

[9] F. P. Junqueira, I. Kelly, and B. Reed, "Durability with bookkeeper," ACM SIGOPS operating systems review, vol. 47, no. 1, pp. 9–15, 2013.

[10] "Apache bookkeeper." https://bookkeeper.apache.org, 2023.

[11] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: wait-free coordination for internet-scale systems.," in USENIX ATC'10, vol. 8, 2010.

[12] "Apache zookeeper." https://zookeeper.apache.org, 2023.

[13] "Pravega - state synchronizer javadoc." https://cncf.pravega.io/docs/latest/javadoc/clients/io/pravega/client/state/StateSynchronizer.html, 2023.

[14] Z. Jia and E. Witchel, "Boki: Stateful serverless computing with shared logs," in Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, pp. 691–707, 2021.

[15] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," in 33rd ACM Transactions on Computer Systems (TOCS), ACM, 2015.

[16] "Intel software guard extensions developer guide." https://software.intel.com/en-us/sgx-sdk/documentation, 2014.

[17] s. Jain, "File system in user space example," Jul 2019.

[18] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," Linux J., vol. 2014, mar 2014.

[19] L. Calcote and Z. Butcher, Istio: Up and Running: Using a Service Mesh to Connect, Secure, Control, and Observe. O'Reilly Media, 2019.

[20] F. Gregor, W. Ozga, S. Vaucher, R. Pires, S. Arnautov, A. Martin, V. Schiavoni, P. Felber, C. Fetzer, et al., "Trust management as a service: Enabling trusted execution in the face of byzantine stakeholders," in 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 502–514, IEEE, 2020.

[21] J. Singh, J. Cobbe, D. L. Quoc, and Z. Tarkhani, "Enclaves in the clouds: Legal considerations and broader implications," Communications of the ACM, vol. 64, no. 5, pp. 42–51, 2021.

[22] G. Reina, A. Gruzdev, P. Foley, O. Perepelkina, M. Sharma, I. Davidyuk, I. Trushkin, M. Radionov, A. Mokrov, D. Agapov, et al., "Openfl: An open-source framework for federated learning. arxiv 2021," Google Scholar.

[23] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," A Practical Guide, 1st Ed., Cham: Springer International Publishing, vol. 10, no. 3152676, pp. 10–5555, 2017.

[24] A. Call, J. Polo, D. Carrera, F. Guim, and S. Sen, "Disaggregating non-volatile memory for throughput-oriented genomics workloads," in Euro-Par 2018: Parallel Processing Workshops, (Cham), pp. 613–625, Springer International Publishing, 01 2019.

[25] A. Call, J. Polo, and D. Carrera, "Workload-aware placement strategies to leverage disaggregated resources in the datacenter," IEEE Systems Journal, vol. 16, no. 1, pp. 1697–1708, 2022.

[26] A. Nestorov, J. Berral, C. Misale, C. Wang, D. Carrera, and A. Youssef, "Floki: a proactive data forwarding system for direct inter-function communication for serverless workflows," in ACM/IFIP International Middleware Conference, p. 13, Association for Computing Machinery (ACM), 11 2022.

[27] J. L. Berral, C. Wang, and A. Youssef, "AI4DL: Mining behaviors of deep learning workloads for resource management," in 12th USENIX Workshop HotCloud, 2020.

[28] "Pubmed." `https://www.ncbi.nlm.nih.gov/pubmed/`. Accessed: May 30, 2023.

[29] "Sequence read archive (sra)." `https://www.ncbi.nlm.nih.gov/sra`. Accessed: May 30, 2023.

[30] M. Carstens, F. M. Rinner, S. Bodenstedt, A. C. Jenke, J. Weitz, M. Distler, S. Speidel, and F. R. Kolbinger, "The dresden surgical anatomy dataset for abdominal organ segmentation in surgical data science," Scientific Data, vol. 10, no. 1, pp. 1–8, 2023.

[31] M. Wagner, B.-P. Müller-Stich, A. Kisilenko, D. Tran, P. Heger, L. Mündermann, D. M. Lubotsky, B. Müller, T. Davitashvili, M. Capek, et al., "Comparative validation of machine learning algorithms for surgical workflow and skill analysis with the heichole benchmark," Medical Image Analysis, vol. 86, p. 102770, 2023.

[32] A. P. Twinanda, S. Shehata, D. Mutter, J. Marescaux, M. De Mathelin, and N. Padoy, "Endonet: a deep architecture for recognition tasks on laparoscopic videos," IEEE transactions on medical imaging, vol. 36, no. 1, pp. 86–97, 2016.

[33] R. Patro, G. Duggal, M. I. Love, R. A. Irizarry, and C. Kingsford, "Salmon provides fast and bias-aware quantification of transcript expression," Nature methods, vol. 14, no. 4, pp. 417–419, 2017.

[34] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. R. Gingeras, "Star: ultrafast universal rna-seq aligner," Bioinformatics, vol. 29, no. 1, pp. 15–21, 2013.

[35] A. Dobin, "STAR GitHub repository." https://github.com/alexdobin/STAR, 2023.

[36] NCBI, "Sratoolkit GitHub repository." https://github.com/ncbi/sra-tools, 2023.

[37] AWS, "AWS Batch on AWS Fargate documentation." https://docs.aws.amazon.com/batch/latest/userguid 2022.

[38] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," arXiv preprint arXiv:1602.05629, 2017.

[39] A. Grafberger, M. Chadha, A. Jindal, J. Gu, and M. Gerndt, "Fedless: Secure and scalable federated learning using serverless computing," in 2021 IEEE International Conference on Big Data (Big Data), pp. 164–173, IEEE, 2021.

[40] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," arXiv preprint arXiv:1412.6572, 2014.

[41] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," arXiv preprint arXiv:1206.6389, 2012.

[42] B. McCall, "What does the gdpr mean for the medical community?," The Lancet, vol. 391, no. 10127, pp. 1249–1250, 2018.

[43] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," IEEE signal processing magazine, vol. 37, no. 3, pp. 50–60, 2020.

[44] A. Mohammed Yakubu and Y.-P. P. Chen, "Ensuring privacy and security of genomic data and functionalities," Briefings in bioinformatics, vol. 21, no. 2, pp. 511–526, 2020.

[45] F. Molnár-Gábor and J. O. Korbel, "Genomic data sharing in europe is stumbling—could a code of conduct prevent its fall?," EMBO molecular medicine, vol. 12, no. 3, p. e11421, 2020.