Funded by
the European Union

HORIZON EUROPE FRAMEWORK PROGRAMME

CloudSkin

(grant agreement No 101092646)

Adaptive virtualization for AI-enabled Cloud-edge Continuum

D2.1 Experiments, Initial Specifications and Testbed specs

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	52
WP/Task related to this document	WP2 / T2.1
WP/Task responsible	Universitat Rovira i Virgili (URV)
Leader	Marc Sanchez-Artigas (URV)
Technical Manager	Josep LL. Berral (BSC)
Quality Manager	Raúl Gracia (DELL)
Author(s)	Marc Sanchez-Artigas (URV), Josep LL. Berral (BSC), Bernard Metzler & Pascal Spörri (IBM), Raúl Gracia (DELL), Sean Ahearne (DELL), Carlos Segarra & Peter Pietzuch (IMP), André Martin & Ardhi Putra Pratama Hartono (TUD), María A. Serrano (NRB), Reuben Docea (NCT), Theodore Alexandrov (EMBL), José Miguel García (ALT), Enrique Chirivella-Perez & Philippe Andrieux & Daniel Ramírez (KIO)
Partner(s) Contributing	All partners
Document ID	CloudSkin_D2.1_Public.pdf
Abstract	Description of use case scenarios, experiments and benchmarking framework. Initial specifications of the architecture and of the Cloud-edge testbed with the enumeration of required services in the continuum.
Keywords	Architecture; Use Cases; Benchmarking; Testbed; Continuum; AI; Orchestration; WebAssembly; Ephemeral Storage; Mobile Computing; Streaming; Metabolomics; Data Space

History of changes

Version	Date	Author	Summary of changes
0.1	01-05-2023	Marc Sanchez-Artigas	Structure of the document.
0.2	12-05-2023	María A. Serrano	Contribute to the orchestration layer with NearbyOne.
0.21	12-05-2023	José Miguel García	Initial Contribution to the agriculture use case.
0.3	19-05-2023	Carlos Segarra, Peter Pietzuch	Contribute to the universal execution layer with WebAssembly.
0.4	06-06-2023	Marc Sanchez-Artigas, André Martin	Contribute to the global architecture description and Confidential execution.
0.45	08-06-2023	Bernard Metzler, Pascal Spörri	Contribute ephemeral storage to the architecture.
0.5	10-06-2023	Raúl Gracia-Tinedo	Contribution of text describing Pravega and its integration.
0.55	12-06-2023	José Miguel García	Contribution of text describing Agricultural Use Case.
0.6	15-06-2023	Reuben Docea, Sean Ahearne	Contribute NCT testbed description for video analytics.
0.7	16-06-2023	Enrique Chirivella-Perez, Philippe Andrieux, Daniel Ramírez	Contribute KIO Networks testbed description.
0.8	22-06-2023	Theodore Alexandrov	Updating the experiments of the metabolomics use case.
0.9	23-06-2023	Josep Lluis Berral	Contribute to the architecture and learning plane description, plus the mobility use case.
0.95	26-06-2023	Ardhi Putra Pratama Hartono	Adding architectural details on Confidential Execution with TEE.
1.0	28-06-2023	Marc Sanchez-Artigas	Updating on the agriculture use case. Updating on the global architecture and envisioned integration. Homogenization of terms and document format. Final version.

Table of Contents

List of Abbreviations and Acronyms

AEMET	Agencia Estatal de Meteorologia (State Meteorological Agency (Spain))
AI	Artificial Intelligence
AoT	Ahead-of-Time
API	Application Programming Interface
AR	Augmented Reality
AWS	Amazon Web Services
C-Cell	Cloud-edge Cell
CAS	Computer-Assisted Surgery
CC	Creative Commons
CFI	Control Flow Integrity
CSV	Comma-Separated Values
DHCP	Dynamic Host Configuration
DL	Deep Learning
DOI	Digital Object Identifier
ECS	Elastic Container Service
FDR	False Discovery Rate
GEDS	Generic Ephemeral Data Storage
GPU	Graphics Processing Unit
HDFS	Hadoop File System
HPC	High-Performance Computing
HPDA	High-Performance Distributed Analytics
JIT	Just-in-Time
JNI	Java Native Interface
JSON	JavaScript Object Notation
K8s	Kubernetes
KPI	Key Performance Indicator
LTS	Long-Term Storage
ML	Machine Learning
MS	Imaging Mass Spectrometry
NAT	Network Address Translation
NVMe	Non-Volatile Memory express

OLTP	Online Transactional Processing
OS	Operating System
PAT	Port Address Translation
PVA	Predictive Video Analytics
QoS	Quality of Service
ROS	Robot Operating System
S3	Simple Storage Service
SDK	Software Development Kit
SFI	Software Fault Isolation
SGX	Software Guard Extensions
SIAM	Sistema de Informacion Agraria de Murcia (Murcia Agricultural Information System)
SLA	Service Level Agreements
SLO	Service Level Objectives
SSD	Solid-State Drive
TCB	Trusted Computing Base
TEE	Trusted Execution Environment
VM	Virtual Machine
WAL	Write-Ahead Log
Wasm, or WASM	WebAssembly

1 Executive summary

The major aim of the deliverable D2.1 “Experiments, Initial Specifications and Testbed specs” is to provide the initial specifications of the CloudSkin platform, along with the description of use cases, experiments and benchmarking frameworks that are part of the CloudSkin project. First off, this deliverable reports on the main components of the CloudSkin platform and their integration with the different use cases. Second, the use cases provide a detailed description of their requirements, as well as the analysis of their potential integration points with the available components of the architecture. Finally, the deliverable describes the different benchmarking frameworks, their requirements, and the test environments used to perform the experiments.

2 Introduction

As of today, 80% of the data processing and analysis occurs in Cloud data centers, while only 20% of processing occurs at the edge. In addition to the dominance of the European Cloud market by non-EU players, the incipient exploitation of edge resources prevents business processes, decisions, and intelligence to be taken outside of the core IT environment.

With the spirit of curtailing the inter-dependency of the non-EU Cloud providers, the CloudSkin project aims to design a cognitive Cloud continuum platform to fully exploit the available Cloud-edge heterogeneous resources, finding the "sweet spot" between the Cloud and the edge, and smartly adapting to changes in application behavior via AI. To facilitate automatic deployment, mobility and security of services, CloudSkin will build a novel universal container-like execution abstraction to allow the seamless and trustworthy execution of (legacy) applications across the whole Cloud-edge continuum. This new execution abstraction will allow the secure processing of sensible information over untrusted Cloud-edge resources with confidential computing techniques. Finally, CloudSkin will design a high-performance infrastructure for the Cloud continuum, tailored to the short-lived, also bursty, execution of Cloud-edge tasks.

2.1 Main innovations

From the above definition, CloudSkin pursues to build a cognitive Cloud continuum platform with three main innovations:

- **[IN1].** The CloudSkin platform will leverage (novel) AI/ML techniques to optimize workloads, resources, energy, and network traffic in a holistic manner for a rapid adaptation to changes in application behavior and data variability. The key goal will be to build a "Learning Plane" that, in cooperation with the application execution framework [IN2] and the Cloud continuum infrastructure [IN3], can enhance the overall orchestration of Cloud-edge resources. This plane will be the materialization of the cognitive cloud, where decisions on the cloud and the edge are driven by the continuously obtained knowledge and awareness of the computing environment through AI, and particularly, neural networks and statistical learning, taking the challenge of enabling these methods into low-power edge devices.

- **[IN2].** The CloudSkin platform will also help users to achieve "stack identicality" across the continuum, whereby the same (legacy) software stacks running in data centers can seamlessly run at remote edges, and not less important, with a high level of security –a critical requirement when processing data off-premises. This will be achieved with the development of a universal virtualization abstraction built upon WebAssembly (Wasm) [1] and the so-called Trusted Execution Environment (TEE) technologies to protect data while it is in use.

- **[IN3].** CloudSkin will also contribute to prepare the needed infrastructure to integrate the new virtualized execution abstractions into the virtual resource continuum, particularly, for those Cloud-edge applications composed of small tasks with fast data access requirements. The infrastructure will expose the relevant control knobs to enable dynamic reconfiguration of resources as assisted by the AI/ML-based orchestration plane in the CloudSkin platform.

In this sense, the proper validation of CloudSkin, along with the successful demonstration of its impact to the EU at all levels, will require of an exhaustive validation of the main innovations through several benchmarks and more importantly, through representative use cases. In particular, the four use cases of the project belong to different European data spaces, say **5G automotive**; **metabolomics**; **surgery**; and **agriculture**.

The 5G automotive use case as a canonical example. Despite the wide variety of domains, the use cases altogether call for Cloud-edge continuum solutions that remain to be built for one reason or another. For comprehensible illustration, we will briefly visit here one out of the four use cases to put to the fore the generic Cloud-to-edge technologies, tools and platforms needed to support the future hyper-distributed applications in the European economy: the **5G automotive** use case:

More concretely, this use case focuses on a mobility scenario where the optimization of resource provisioning, data movement and application placement are cornerstone elements. This use case will consider two scenarios: a network-bound setting, where the workload and data have to follow a user across the Cloud-edge infrastructure (horizontal service migration across edge servers), and a computing-bound situation, where heavy workload needs to be placed closer to the user as much as possible (vertical service migration across the Cloud continuum). Therefore, this use case not only requires of a smart orchestrator to automatically deal with the placement and migration of tasks across the Cloud-edge infrastructure, but also a universal virtualization abstraction platform that enables the seamless execution of tasks on a wide array of Cloud and embedded devices. This new abstraction must be migratable across different servers and devices in the continuum to effectively support both horizontal and vertical service migration.

Roadmap. The following document provides an overview of the project and the initial specifications of the novel CloudSkin platform. In the following sections, the main components of the architecture will be described along with their specifications and envisioned integration. Next, the use cases will be detailed, with a particular focus on their requirements, datasets and experiments. Likewise, the benchmarking frameworks will be defined for the correct evaluation of each use case.

3 Initial architecture specifications

3.1 Global Architecture

The three main innovations [**IN**$1-3$] will contribute new software components that will be unified in the CloudSkin smart continuum platform through a **layered design**:

- **L3. Orchestration layer.** A fundamental piece of the CloudSkin software stack is its **AI-enabled orchestration layer**. The main goal of this layer is the identification of the best provisioning, placement and partitioning policies and strategies between the Cloud and edge servers, while dynamically fulfilling the changing requirements of applications. Different tasks might have different resource demands or data transfer requirements, either on the edge and the Cloud. To this aim, at the core this layer will lie an innovative **Learning Plane** as the AI-based tool towards "smart" and holistic orchestration of the Cloud-edge continuum. The Learning Plane will be in charge of extracting knowledge from the continuum components, and provide the full software stack with recommendations, predictions and additional inferred information towards decision making and global system optimization. This will include the resource provisioning mechanisms, such as virtualization, containerization and storage in the execution layer, as well as the storage services in the infrastructure layer.

  This layer will leverage and will be leveraged by the available orchestrators in the control plane, feeding the existing orchestration stacks with valuable information to better guide resource provisioning, or optimally partition tasks between the edge devices and the Cloud. Observe that orchestration and management can vary significantly depending on the domain at hand, making different stacks a better option in each case. To wit, Kubernetes (K8s) [2] is an ideal orchestrator for Cloud-native applications, while other orchestrators like NearbyONE [3] are specifically tailored for 5G "telco" scenarios. For this reason, the new AI-enabled orchestration layer will be divided into two planes: the control plane and the data plane, in addition to the Learning Plane, to better adapt orchestration decisions to the execution environment.

  The design of the orchestration layer will the include an orchestrator as a main component of the control plane. Orchestrators such as NearbyOne for 5G edge orchestration, K8s for Cloud-edge environments, and Lithops [4, 5] towards serverless architectures, among others to be studied and proposed. The control plane will be responsible for the service on-boarding and lifecycle management of applications at a global and local scales, across the Cloud continuum consisting of multiple heterogeneous sites. Further, the data plane will include components responsible for collecting and managing metrics and telemetry, extracting knowledge from both the underlying infrastructure and the decision-making systems using specialized tools such as `Prometheus`[1].

  Overall, this architecture will enable flexibility on services composition under the same design, avoiding locking on specific technologies, but allowing each case to evaluate the relative trade-offs on each technology, environment and paradigm, relative to their objectives such as cost, performance, flexibility or scalability. A more detailed description of the orchestration layer can be found in deliverable **D5.1**.

- **L2. Execution layer.** Tapping into the CloudSkin platform, application developers will be able to implement general solutions capable of spanning the whole Cloud-edge continuum. This will be possible because CloudSkin will provide a **universal** and **adaptive virtualization layer**. That is, "universal" because will offer a lightweight execution environment with a similar (or even "identical") software interface, allowing unmodified code to be run in any machine in the system. And "adaptive" because will be able to transparently leverage hardware-based acceleration and/or isolation support when available, for example, with TEEs to facilitate the confidential processing of sensitive data off-premises such as Intel SGX [6].

[1] `https://prometheus.io`

The execution layer will be built upon WebAssembly [1], because it has the sufficient generality to support continuum applications, as well as a superior lightness compared to containers [7], resulting in a powerful tool to run multi-platform software with non-significant performance degradation and small memory footprints. This new execution unit will be called "**Cloud-edge Cell**", or C-Cell for short.

It must be noted that for some use cases, where certain computations do not need to migrate across the continuum, non-WebAssembly software stacks (e.g., Apache Spark) that are complex to compile to WebAssembly will be leveraged **as is** for better performance.

The software architecture for C-Cells is extensively described in deliverable **D4.1**. But for a brief overview, we must say that C-Cells will be built from the lessons learned in the development of `Faaslets` [7], a lightweight isolation mechanism based on WebAssembly. More concretely, WebAssembly's memory safety guarantees, namely **Software Fault Isolation (SFI)**, will enable C-Cells to execute side-by-side in a single instance of the runtime. This design will provide a large amount of flexibility for the cloud continuum operation because, for instance, it will allow Wasm-sandboxed code from different tenants to run within the same container, VM etc., with equivalent semantics to threads and processes, as well as transparently moving it around the continuum.

Of course, all the above will be realized while ensuring a lower invocation latency and memory footprint. To span the whole continuum, great effort will be put on supporting heterogeneous architectures and instruction sets. As of today, not all WebAssembly execution modes, namely, *interpreted*, *Ahead-of-Time* (AoT) compiled, and *Just-in-Time* (JIT) compiled are supported in all architectures. Its design will be driven towards making C-Cells a truly universal execution unit for the continuum. Further, the C-Cells' runtime will provide the necessary hooks to track C-Cells' lifeycle, react to migrations, etc., as instructed by the AI-enabled orchestration layer.

- L1. **Infrastructure layer.** The modern lightweight virtualization technology developed in the execution layer will enable compute units to scale up or down in milliseconds, reporting rapid responses to data consumers. However, breaking the monolith applications into small tasks can lead to high overheads, in particular for large bursts of tasks running for a few ms. Routine I/O operations such as the ingestion of a sequence of compressed video frames, or the maintenance of a shared state between tasks, can be definite showstoppers for the efficient execution of edge-to-cloud bursts if not properly handled. That is, an efficient execution abstraction for the continuum such as that of L2 with very low startup times can render useless if I/O operations are comparatively slow. For instance, for short-lived tasks, resorting to disk-based storage with I/O operations amounting to 5-10 ms may be a too high penalty to pay.

  For the above reasons, the key role of this will layer will be to develop efficient cloud-to-edge storage services for efficiently managing ephemeral data. Altogether with an efficient execution layer in the continuum, we envision that these services will become more than ever determining factors to fuel the continuum market in the coming years.

  A key requirement in this layer is not falling in the trap of overengineering the solution with unnecessary features that make the system lose generality. Capitalizing on a long experience in building high-performance and ephemeral storage such as Apache Crail [8] and Pocket [9], this layer will up our efforts to develop a new ephemeral storage service called "**Generic Ephemeral Data Store**", or GEDS for short, to provide advanced support for data management across the continuum.

  By now, clients to the data store can join the GEDS service by loading the `libgeds` shared library. This library provides several functions to manipulate ephemeral data objects. For instance, depending on how an object has been opened, GEDS may request the data over the network, or keep a copy of the data locally in memory for fast reuse. GEDS is currently under active development, and it is expected new functionality to come out in the new deliverables, such as a publish-subscribe mechanism to enable clients to track the I/O operations performed on objects of interest.

Moreover, we will integrate a **streaming storage fabric** for use-cases with stringent low-latency streaming requirements, such as real-time video analytics. To this end, we rely on Pravega [10] as a streaming storage building block. During this project, we will extend the functionality of Pravega to adapt to heterogeneous infrastructure scenarios, as well as to become an efficient data source for C-Cells' computations.

To summarize, the CloudSkin layered architecture will be built on new enabling technologies contributing to core layers of the cloud continuum, which are:

- Smart management and orchestration functionalities (layer L3);
- Adaptive virtualization and universal execution environment (layer L2); and
- Optimized management of ephemeral data (layer L1).

From a bird's eye view, the layered architecture is depicted in Fig.1.

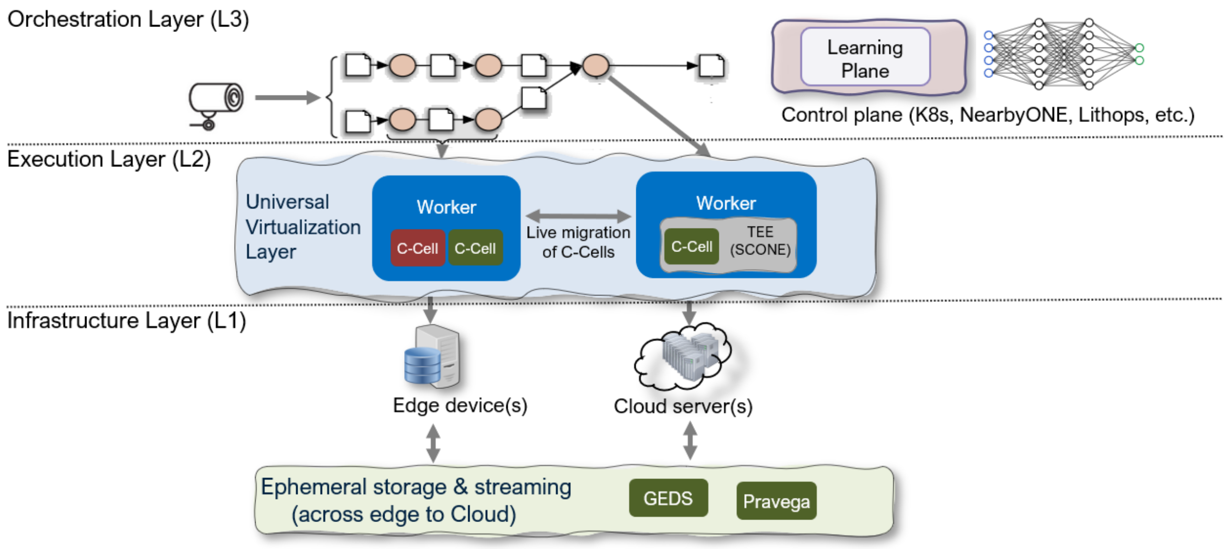

Orchestration Layer (L3)
Learning Plane
Control plane (K8s, NearbyONE, Lithops, etc.)
Execution Layer (L2)
Universal Virtualization Layer
Worker
C-Cell
C-Cell
Live migration of C-Cells
Worker
C-Cell
TEE (SCONE)
Infrastructure Layer (L1)
Edge device(s)
Cloud server(s)
Ephemeral storage & streaming (across edge to Cloud)
GEDS
Pravega

Figure 1: Layered Architecture for the CloudSkin platform.

As one of the most common architectures, a layered approach presents many benefits. First off, it provides a clear separation of concerns, where each software layer performs a specific role within the continuum. But also, it favours change isolation, i.e., future changes in one layer specification should not affect the rest of the layers. As an on-going research project, a layered design will make it easier to support an iterative methodology with short design-prototyping-validation cycles for the project.

A canonical example of a CloudSkin platform deployment is depicted in Fig. 2. As can be seen in this figure, Kubernetes will act as the control plane for our smart orchestrator, scaling up and down resources based on the AI-based Learning Plane. Cloud-edge cells, or C-Cells, whether equipped with modern TEEs such as Intel SGX or not, will run within worker instances as pods. Compared to containers, C-Cell workers will provide higher efficiency because a higher number of C-Cells will be able to run on the same virtual machines, or on hardware where containers are too heavyweight. Finally, the ephemeral storage service built upon GEDS will be used as a micro-latency storage system to exchange data between small tasks of a few milliseconds duration, where otherwise collective communication patterns such as broadcast will be problematic due to the huge number of network connections to be established among task executors.

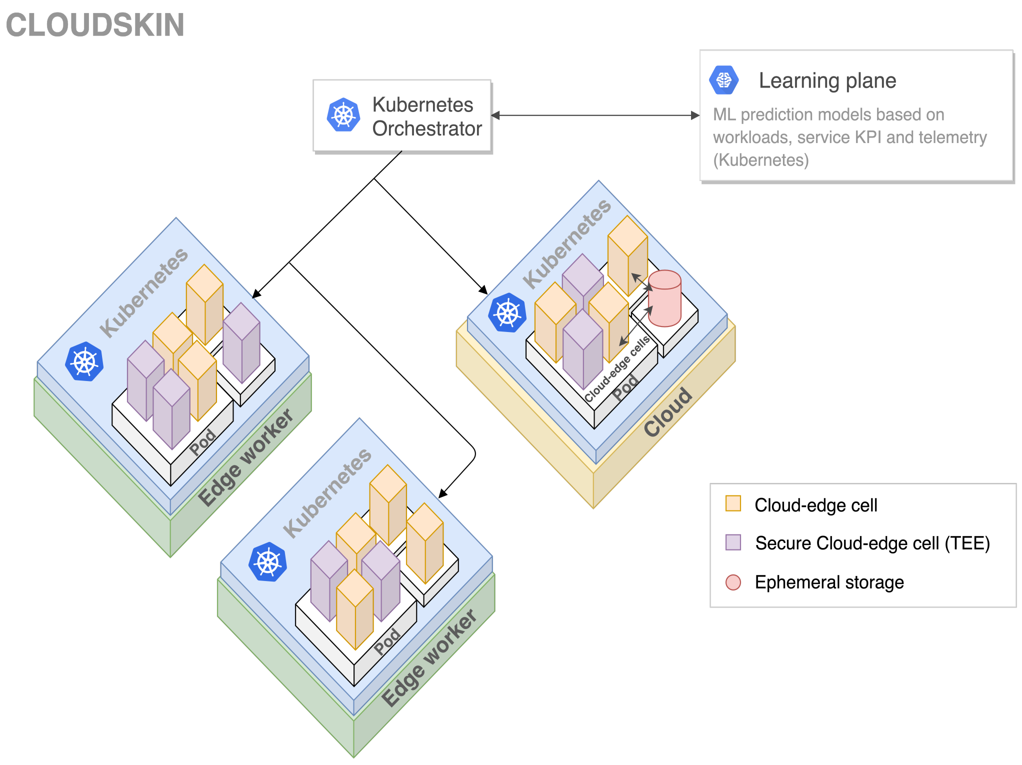


Figure 2: Example of a CloudSkin platform deployment.

It is important to note that CloudSkin platform will not be tied to a concrete orchestration service. As discussed at the beginning of this section, the specific orchestration service will be abstracted as much as possible from the architecture (e.g., through `shims` or client APIs) to maximize the scenarios where the platform can be deployed. The use of a meta-orchestrator, that is, an orchestrator that works like any other Cloud-edge orchestrator, but instead of managing resources, it orchestrates the application through other low-level orchestrators, could be an alternative solution.

Software components. Table 1 provides the complete list of the software technologies that will be integrated into the CloudSkin platform. For each software tool, the table provides a short description of it and indicates the main software layer where it belongs, namely layer L1, or the infrastructure layer; layer L2, or the continuum execution layer; and layer L3; or the AI-enabled orchestration layer. As listed in this table, the CloudSkin software platform will be based on components delivered by the partners, most of them **open-sourced**, or on well-known **open-source** software components.

As expected, the tools contributed by the partners will be reworked to provide the functionality needed to meet the project goals, integrating them in a standard off-the-shelf DevOps environment as required by the use cases. One example of this is the new C-Cell abstraction, which will be built by reshaping Faasm [7], a high-performance stateful serverless runtime, as stated in deliverable D4.1. In this regard, the project represents a great opportunity for partners to add the functionality needed to support Cloud-edge applications into their portfolio offerings (e.g., NRB will integrate the project advances into its flagship 5G orchestration service: NearbyOne [3]).

In what follows, we will provide an overview of the architecture of each layer in isolation. A first complete iteration of each layer can be found on the following public deliverables:

- D5.1 “Design and early prototype of CloudSkin Learning Plane” (Layer L3).
- D4.1 “Initial prototype for Cloud-edge cells” (Layer L2).
- D3.1 “Early release of Ephemeral Data Store” (Layer L1).

Table 1: Software technologies for CloudSkin.

Name	License (Owner)	Software Layer	Short Description
Kubernetes; Knative	Open-source (CNCF)	L3 – Orchestration	Container orchestration systems for automating software deployment, scaling, and management.
SCONE	Community Edition (SCONTAIN)	L2 – Execution	Open source confidential computing platform that supports the execution of sensitive applications with TEE technology inside of containers.
Faasm	Open-source (Apache)	L2 – Execution	High-performance stateful serverless runtime that Faasm combines software fault isolation from WebAssembly with standard Linux tooling.
Lithops	Open-source (Apache)	L3 – Orchestration	Lithops is a Python multi-Cloud serverless data processing framework. Lithops offers an extensible architecture with compute and storage backends for major Cloud providers and open source container platforms.
TensorFlow; Pytorch	Open source (Apache; BSD)	L3 – Orchestration	Frameworks used for ML and AI development, mainly focused on training and inference of deep neural networks.
OSM; Nearby-One	Open-source (ETSI); Propietary license (NRB)	L3 – Orchestration	O-RAN aligned orchestration tools towards open and fully interoperable mobile.
GEDS	Open-source (Apache)	L1 – Infrastructure	Fast, distributed and multi-tiered data store that is being designed specifically for managing ephemeral data.
Pravega	Open-source (Apache)	L1 – Infrastructure	Pravega is an open source distributed streaming storage service. A Pravega stream stores unbounded parallel sequences of bytes in a durable, elastic and consistent manner while providing unbeatable performance and automatically tiering data to scale-out storage.

3.1.1 Layer L3 – AI-enabled orchestration layer

Given the heterogeneity and non-reliability of Cloud-Edge infrastructures, the leverage of AI allows refining orchestration decisions and raising the agility of the orchestration software stack to properly to changing conditions. CloudSkin proposes the construction of a "Learning Plane", connected to the control and data planes, in order to manage the knowledge obtained from the different components of the infrastructure, but also from the decision-making systems. To this aim, here we specify the components for the three planes: learning, control and data planes, responsible for provisioning resources, application placement and data passing, as seen in Figure 3, among other key activities. Next steps on the design will be to select the specific technologies to implement the functionalities for each component, according to the use case specifications and details. This will be done by leveraging the advances on partner projects such as NearData (HORIZON GA.101092644) on data-connector standardization.

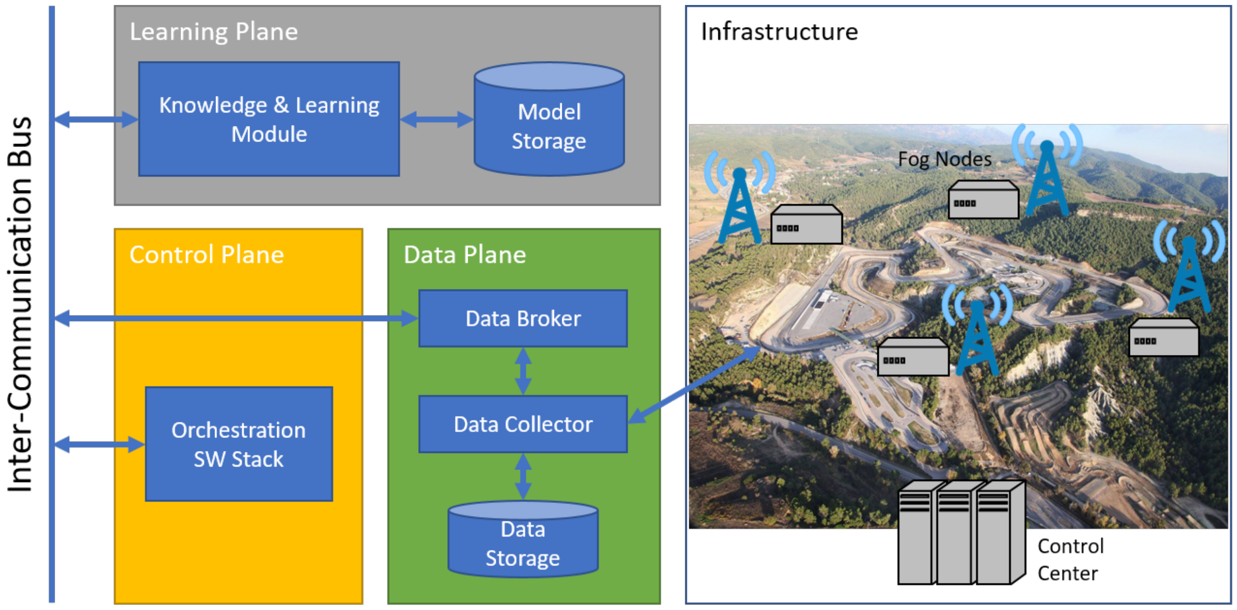


Figure 3: AI-enabled Orchestration Architecture, with the Use Case 4.1 as an example.

Requirements. CloudSkin imposes a series requirements on the design of orchestration layer, such as:

- **Smartness**: Decision-making for management, namely workload provisioning and placement, must be done through "smart" methods. These methods must be fast and adaptive, avoiding both classic exhaustive approaches and ad-hoc methodologies. For this, the orchestration layer must leverage modern AI techniques (i.e., Machine Learning and similar heuristics) to make recommendations and decisions agile, while making them updateable.
- **Autonomy**: In hyper-distributed systems, orchestrating the whole system is totally unfeasible. Classic orchestration systems can cover very large computing systems, but when moving to the edge, such systems can become fragmented, unreachable or unavailable. For these reasons, the orchestration layer must have into account the possibility of managing localized systems as a whole autonomous system, allowing higher abstraction planes (e.g., the Learning Plane for models or the data plane for storage) to federate components and regions of the edge.
- **Lightweight**: Methods for decision-making must remain simple in the continuum. Complex methods require computational resources that might not be available in edge systems such as GPUs. Lightness is thus particularly relevant for effectively pushing management actions near

data. This involves schedulers, monitoring tools, analytics, ML methods, and even hypervisors. For this, the orchestration layer must deploy tools and methods that are not computationally expensive, easy to deploy, and easy to trace. Keeping explainability of decisions also plays an important role, even at the cost of a minimum loss of accuracy in decision-making.

- **Heterogeneity**: Workload and resources condition the policies for provisioning and placement, requiring adaptability. Therefore, the orchestration layer must be aware of the heterogeneity of the environment for infrastructures and deployed platforms, considering the diversity of resources and workloads.

Learning Plane Design. The Learning Plane is composed by a module containing ML prediction and recommendation methods, trained from existing examples on the architecture functioning (i.e. collected monitoring traces from application performance, resource usage, provided QoS, etc.), to be queried by the control plane components when decisions on provisioning or placement must be made, or when QoS must be enforced. The initial design and implementation of the module includes the ML models for performance prediction and forecasting, a service process listening to queries from the control plane requesting recommendation, and a bus/channel connecting with the control plane components (to receive queries and communicate responses) also with the data plane components (retrieving collected information from the infrastructure, towards modelling and prediction). The initial approach for the Learning Plane components and architecture is as follows:

- **Modeling Algorithms**: The models will be trained from telemetry data (e.g., retrieved from the monitoring mechanisms in the data plane). Telemetry is considered data to be managed by the regular Cloud-edge orchestration mechanisms. The component will create performance models, capable to predict the status of the infrastructure and application of interest, also to forecast its status, to produce a recommendation at demand by the control plane.

- **Learning Storage**: The models will be stored, either centralized or distributed. And they will be loaded on demand as required by the prediction service. The objective of enabling distribution is to prepare the design towards decentralization in case of infrastructure disconnection or in case of federated management.

- **Prediction Service**: It will listen to queries from the control plane, providing recommendations or forecasting that can help in the decision-making process. An API will be defined to provide all the functionalities required by the use cases. The potential technologies to implement such service are gRPC, for remotely calling implemented ML prediction functions, or equivalent HTTP-based application service engines (e.g. Apache Flask).

- **Data Retriever**: The data retriever will fetch and pre-process the required data and information from the data plane (i.e., telemetry from the data broker) and control plane (e.g., information provided by the orchestrator, as parameters or planning information). It will be implemented towards the data broker and orchestrator available APIs.

The initial architecture concept is shown in Figure 4.

Control Plane Design. The control plane is, needless to say, composed by the software stack for Cloud-edge orchestration. It is responsible for maintaining the on-boarding along with the life-cycle of the applications across the infrastructure, both globally or locally. The orchestration must be able to provide the services of:

- **Monitoring**, retrieved from the data plane;

- **Analysis**, retrieved from the Learning Plane;

- **Planning**, including scheduling and policy enforcing;

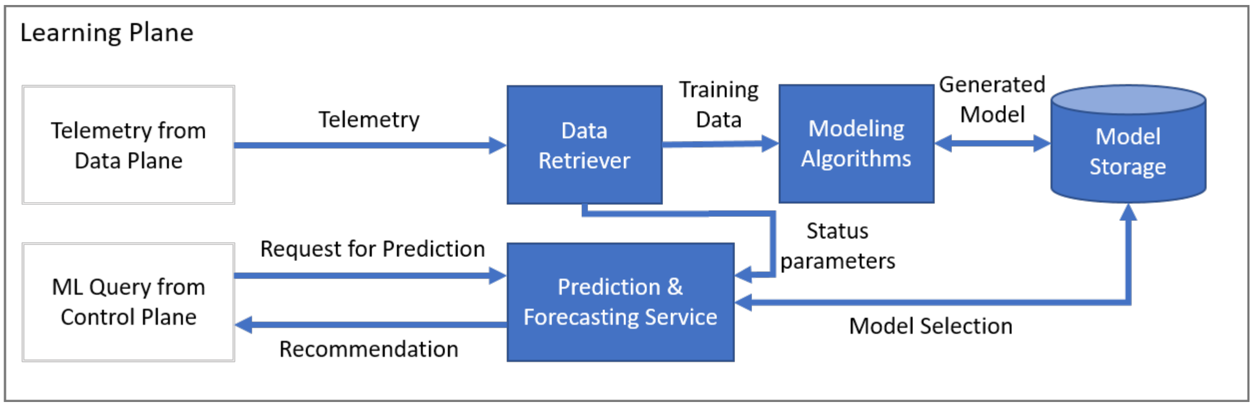


Figure 4: Learning Plane initial schema and pipelining.

- **Execution**, i.e., workload provisioning and placement.

The requirements for this design are:

- **Telemetry Retrieval**: Advanced orchestration mechanisms for Cloud-edge management need specific telemetry and fine-grain resource control mechanisms, obtained from both the data and Learning planes.
- **Policy Enforcer**: The global planner function is responsible for deciding where to run a given services, or app, how to assess their KPIs with respect to established Service Level Objectives (SLOs), and how to set up the services and the infrastructure to meet user expectations, a.k.a. provisioning.

The selected orchestration stacks (e.g., NearbyONE, Kubernetes, Lithops, etc...) will be chosen by their capabilities of such functions in the given specific scenarios and runtime paradigms where to be deployed.

Data Plane Design. The data plane is responsible for the transmission of management data from the different planes. In the specific case for the orchestration layer, the data plane is in charge to retrieve the telemetry from the computing environment, collecting metrics from the infrastructure, platform and even models from the Learning Plane. The data plane components are the following ones:

- **Data Broker**: The component is in charge of processing, aggregating and propagating data from a variety of sources to the Learning and control planes.
- **Data Collector**: The component is in charge of extracting data, periodically or event-based, from one or more upstream data sources in the most automatic manner.
- **Data Storage**: The component is in charge of enabling long-term data preservation and storage.

The data plane will be powered by standard open-source components like `Prometheus` [11], to gather data from the different components of the infrastructure, and `Thanos` [12] for long-term data aggregation and storage.

3.1.2 Layer L2 – Universal execution layer with WebAssembly

One of the major CloudSkin aims is to build an execution environment with adaptive virtualization for the AI-enabled cloud-edge continuum. A key piece of this execution environment is its execution unit, we name it "**Cloud-edge cell**", or **C-Cell** for short.

Requirements. CloudSkin imposes stringent requirements on the design of C-Cells. These are:

- **Cloud-edge support**, which means that C-Cells need to support the seamless execution of user code in a variety of architectures and instructions sets, and with dissimilar requirements in terms of latency and cost. Existing virtualization techniques that have been widely adopted in cloud environments, namely virtual machines (VMs) and containers, are often ill-suited: VMs are typically too heavy-weight entities that rely on hardware virtualization support, which is unavailable on many resource-constrained edge hardware architectures.

- **Continuum support**, which means that C-Cell execution must be able to be interrupted and transferred from one host to another across the heterogeneous Cloud-edge continuum with no (or little) disruption to the application execution. Within the continuum, task offloading is a very widely used approach to better utilize the diverse computation resources, both on the edge side and cloud side, which contributes to an extended execution pipeline that must be supported by C-Cells.

- **Adaptive virtualization**, which means that optionally and transparently, depending on the data being processed, C-Cells must support hardware-based acceleration and/or isolation, for example with TEEs such as Intel SGX [6]. This property is very important to build a unified technology able to support the ever-changing demands of businesses.

- **Runtime environment**, which means that the design of an execution unit for the continuum needs to be accompanied by the design of a runtime environment capable of executing these units. The C-Cells together with the runtime environment will make up the execution layer for the CloudSkin platform. The runtime environment is covered in detail in deliverable D4.1.

Design. The design of the execution layer builds upon the learnings from Faasm [7], a distributed serverless runtime. Faasm introduced **Faaslets**, a novel lightweight isolation mechanism based on the memory safety and control-flow integrity (CFI) guarantees provided by WebAssembly. Faaslets are a good starting point for C-Cells as they are lightweight and portable. However, C-Cells are not a mere recast of Faaslets since their internals will need to be significantly modified in several ways to accommodate the above four requirements:

- First, Faaslets only support being executed in x86-64 servers. C-Cells, on the contrary, will need to be transparently executed in different architectures such as RISC-V and AArch64, to name a few.

- Second, Faaslets do not support live migrations. Despite research in live migration is abundant, C-Cells faces a unique difficulty: live migration needs to happen between heterogeneous hosts, potentially running different instruction sets.

- Third, an application in the CloudSkin platform is expected to be scattered across the Cloud-edge continuum. We thus need a way to group C-Cells belonging to the same application, we call C-Cells groups **Tissues**. C-Cells in the same Tissue should be able to message each other and/or share memory. This is not as simple as it seems, since for C-Cells that communicate via message passing or make use of shared memory (i.e., using some sort of multi-processing or multi-threading API), special care must be taken when checkpointing their state, in order to not lose any in-flight messages or shared state.

- Finally, Faasm has a simple distributed-state scheduler, similar to Omega [13]. Requests are randomly routed to a worker instance. If a function need to be scaled up, the worker will spawn Faalets locally, and scale-out to other workers if local resources are exhausted. To integrate with CloudSkin's requirements we need a centralized scheduler that can integrate with the Learning Plane.

We present the envisioned architecture for CloudSkin's execution layer in Figure 5. The biggest differences with Faasm's architecture [7] are: 1.- The compute infrastructure is spread across different

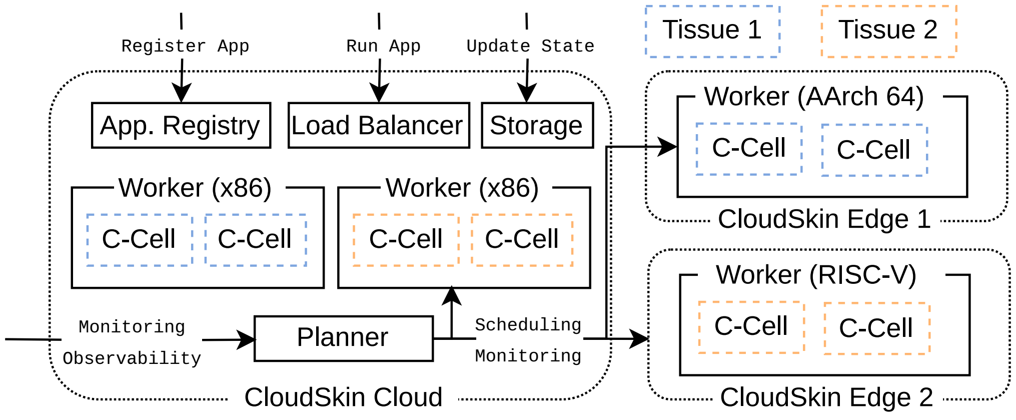


Figure 5: High level diagram of the CloudSkin's execution layer for the Cloud-edge continuum. The execution unit in CloudSkin are C-Cells, which are groupped in Tissues and execute WebAssembly bytecode.

physical locations, 2.- C-Cells can execute in different hardware resources, 3.- C-Cells can be grouped in Tissues to send messages and/or share memory, and 4.- Scheduling is centralized with a new component called **Planner**, that will integrate with the learning plane.

Confidential Execution Layer with TEEs. In order to ensure confidentiality and integrity for the execution layer, we envision the use of SCONE [14], a framework that facilitates the use of Trusted Execution Environemnts (TEE) [15] such as Intel SGX [6]. The objective here is to protect data at rest, i.e., stored on persistent storage; data in use, i.e., while being processed; as well as data in transit where it is exchanged/transferred between hosts and C-Cells.

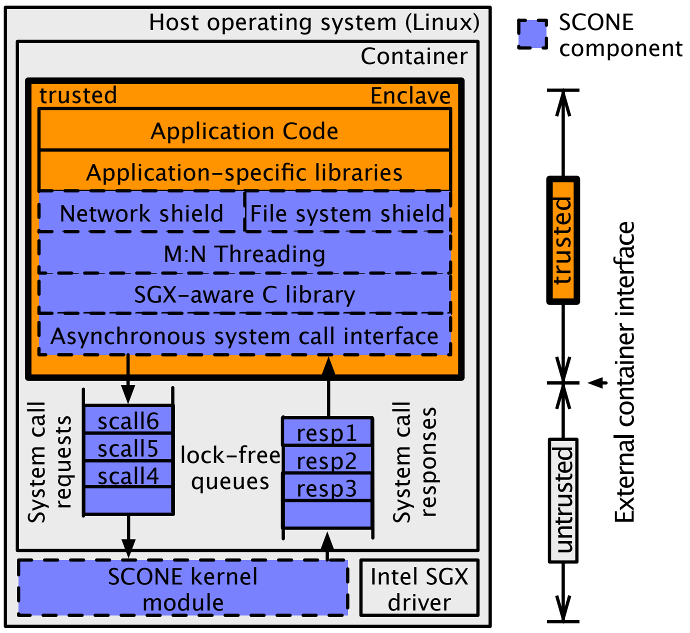


Figure 6: SCONE [14] Architecture.

Today, to make use of TEEs, developers have to be familiar with the different hardware capabilities and utilize the provided SDKs to enable their existing source code, either specific functions or the whole executable, to run in such trusted compartments, namely, enclaves. Our proposal is to alleviate those challenges and add more capabilities regarding confidential execution. Figure 6 shows the architecture of our approach using the SCONE platform.

Within CloudSkin, we propose and envision a transparent approach where native applications such as WebAssembly code are transformed into confidential assets. Such an approach is possible thanks to the *SGX-aware C library*, which is also compatible with various programming languages. This approach will be furthermore complemented by an automatic configuration approach to ensure that processes and services exchange data in a secure fashion, including mutual authentication to establish trust between the communicating parties. Finally, the *network shield* will be used to protect all external communication by encrypting it, regardless of the application's specification.

Also, we introduce transparent file system encryption, which neither needs any modification of the existing code-base nor the use of explicit overlay encryption techniques such as that provided by user-space-based filesystems (e.g., Fuse [16]). This approach, leveraged by the *file system shield*, will provide transparent encryption and decryption of data within the trusted compartment, thereby providing the desired confidentiality and integrity properties needed for C-Cells.

The execution layer, including the application, is now part of the Trusted Computing Base (TCB) thanks to the TEE architecture. SCONE provides an external interface that not only transparently translates all system call invocations but also checks all returns to the application to ensure integrity and confidentiality. SCONE adopts an *M:N threading* model, which means *M* application threads are mapped to *N* threads on the OS. Moreover, by having it asynchronous and lock-free, it maintains high performance despite being executed in an enclave.

3.1.3 Layer L1 – High-performance ephemeral storage and streaming storage fabric

Although the development of the Learning Plane (layer L3) can facilitate the real-time workload-aware revision of the Cloud-edge partition strategy, and handle unpredictable application patterns and dynamics, the materialization of the cognitive cloud cannot thereby be disassociated from the infrastructure that restrains its "radius of action".

One representative example of how the properties of the infrastructure constrain the Cloud-edge cooperation is when a burst of tasks occurs on the edge side. Examples of bursty workloads can be found in traffic roads during rush hour, real-time video analytics [17], or computer-assisted surgery, one of the project use cases, where video data must be analyzed in real-time and with low latency. When a burst occurs at an edge server, a large number of tasks may be accumulated in a short time period. Such tasks may be exfiltrated to the cloud or migrated among edge servers as indicated by the Learning Plane. While the minimalistic virtualization of C-cells will allow to start up such bursts very quickly, access and exchange of data between small tasks have to be at least one to two orders of magnitude faster, i.e., microsecond latency, to keep up. Further, storage must inherit the very same ephemeral nature of tasks to make an optimal use of the infrastructure resources. Bursty applications do not make efficient use of their resources. They require a large resource allocation during bursts, followed by resource underutilization in between.

The main contribution of the infrastructure layer will be the design and implementation of fine granular, elastic and ephemeral base services for the execution of C-Cells.

Requirements. Again, CloudSkin imposes stringent requirements on the design of storage layer. These are:

- **Performance:** In distributed systems, the efficient handling of ephemeral data is critical for the overall performance of workload execution. Only the availability of bare metal infrastructure performance at the storage level will enable the flexible and fine granular decomposition of complex application workflows into sets of parallel and short-lived tasks, so that the CloudSkin platform can efficiently run its intended use cases in a cloud-edge continuum with C-Cells. In this sense, I/O operations will be deliberately made as much simple as possible, targeting sub-millisecond latencies.

- **Elasticity:** In a dynamic edge-to-Cloud continuum, elasticity is vital to strictly ensure that only no extra storage resources are provisioned to hold the working dataset, a feature that clearly distinguishes an ephemeral storage layer from a classic managed services such as a key-value store (e.g., AWS S3). Elasticity can be achieved at multiple levels, such as it occurs in Pravega, which is able to automatically change the parallelism of the individual data streams to handle fluctuations in the data ingestion rate.

- **Multi-tiering:** While storing all the ephemeral data in DRAM is preferred from a performance standpoint, doing so typically is too costly or impossible on low-end edge devices. Hence, an efficient storage platform for temporary data should integrate multiple storage technologies that offer different performance cost trade-offs. Overall, multi-tier storage can be beneficial for several reasons in the continuum. First, it allows to dynamically exceed local storage resource bounds, either accidentally or intended (e.g., by relocating the less frequently used objects of thin edge devices). Second, it enables the integration of a cloud storage service (e.g., HDFS, AWS S3, etc.) for reading input data and writing final output data seamlessly under a common namespace. And third, a "higher" storage tier can be used for checkpointing and recovery of working datasets in case of failure.

- **Object location awareness:** This requirement concurs with the performance rule-of-thumb of keeping data as close as possible to the current task to avoid network data transfers, as well as local data copy operations during read and write access. Since ephemeral data objects only live for tens to hundreds of seconds, migrating this data to re-distribute load when nodes are overloaded may have a high overhead. For such a reason, this property is essential to enable the coordination of data output placement and compute task-scheduling, aiming at exploiting data and compute colocation at the same node.

High-performance ephemeral storage. With the **G**eneric **E**phemeral **D**ata **S**tore, GEDS for short, we will we try to address all of the above requirements. Fortunately, we will not start from scratch and we will tap into the lessons learned during the design of another high-performance ephemeral data store: Apache Crail [18], which exemplifies a design for high-performance, distributed object access and flexible storage multi-tiering.

As with the execution layer, a mere recast of Apache Crail will be unable to meet the stringent requirements of the project, for we add two important features to GEDS, which are not fulfilled by this store. These are:

- A deep integration of communication buffers into the application itself to support, among other things, zero-copy memory-mapped data access to objects.

- The support of new task synchronization primitives such as a publish-subcribe service to allow clients to subscribe to specific data objects and track their lifecycle, i.e., the number of different phases they go through from birth to death. For instance, this will allow triggering activities related to the observed objects, such as the automatic ingestion of ephemeral data arising from a previous stage of data processing. In this way, applications will be able to skip to set up an explicit control flow among stages, while enabling direct object transfers between consecutive tasks.

Figure 7 depicts the overall design of GEDS. At the time of this writing, the data store comprises the following three main components: data storage tiers, clients and a metadata service.

Data plane. Currently, the data plane consists of three distinct storage tiers: 1.- **Tier Zero**, which implements a high-performance ephemeral storage tier, where data objects lie in local node memory, potentially being memory mapped into application buffers; 2.- **Tier 1**, which means a disaggregated storage tier designed to keep data objects in high-performance block storage such as NVMe disks; and 3.- **Persistence Tier**, which is the lowest storage tier, and that it is used to persist into and retrieve

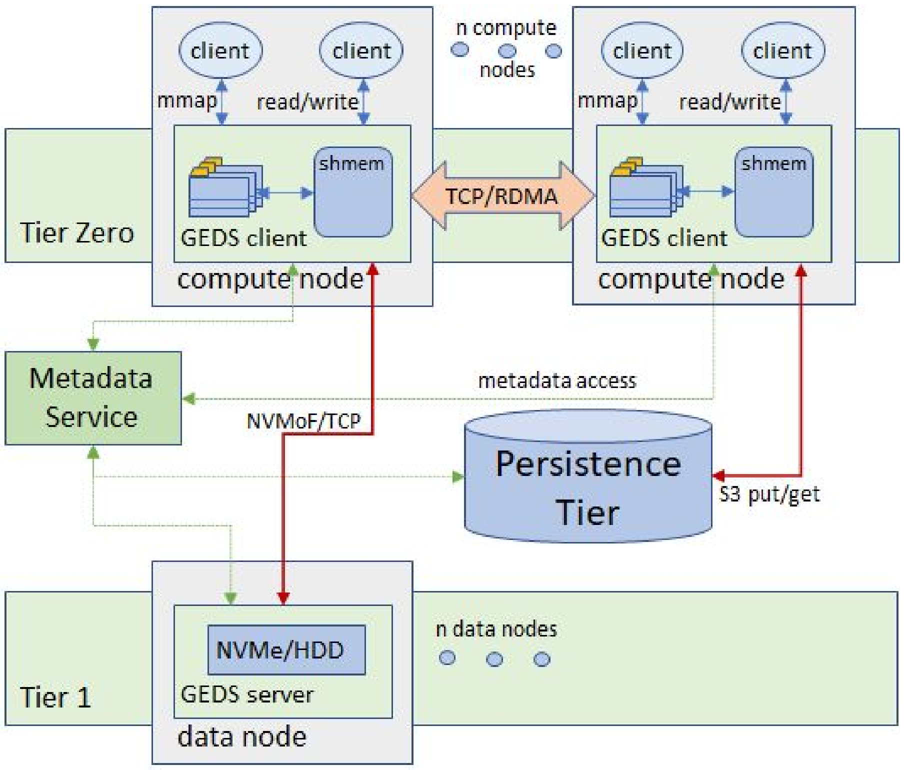


Figure 7: High-level diagram of the GEDS architecture.

data from an available key-value store such as AWS S3. In addition, each GEDS deployment runs (at least) one **metadata server** (MDS), whose main role is to track the location of objects across the servers in the data plane. The metadata includes the information on which tier an object can be found.

Integration with applications. At the time present, clients can join the GEDS service by loading a `libgeds`, a client shared library. After the library has been loaded, the client registers itself with the metadata service, and can start participating in the GEDS service. Interestingly, participation can be passive by simply reading data objects, or active by registering and writing new data objects. Simply put, every time a client opens a file GEDS hands out a `GEDSFile` object. This object ensures that the GEDS service keeps the file allocated in the data plane. From there on, the client can then execute `read` and `write` operations on the `GEDSFile` object. Depending on how this object has been opened, GEDS may request the corresponding data over the network, or just keep a copy of the data locally.

A stable prototype of the GEDS data store with limited functionality is openly available at [19]. The core of GEDS is written in C++. It exports native language bindings for Java via JNI (Java Native Interface) and for Python via `PyBind11` [20].

Streaming storage fabric. Pravega is a distributed, tiered storage system for stream processing. The design of Pravega provides a natural fit for addressing most of the requirements specified above for streaming use-cases, such as performance, elasticity and multi-tiering. Indeed, it is the only system providing the concept of elastic data streams. While exploiting an open-source and mature streaming storage system is an advantage, Pravega needs further innovation to fully address the challenging use-cases of CloudSkin. These are:

- The ability to stretch across heterogeneous infrastructures. Simply put, Pravega was originally designed to run on single-cluster deployments. One important goal during the development of the Project is to equip Pravega with a logical layout that enables stretching a cluster across multiple and heterogeneous infrastructures (edge, core, Cloud), while being able to adapt the instance resources allocated to each infrastructure.

- Exploiting Pravega clients via secure and low-latency C-Cell computations. While Pravega can be used from a wide variety of applications and analytics engines, there are still unknowns when it comes to run secure stream computations via TEEs, or serving cutting-edge compute frameworks such as WebAssembly. Still, overcoming these open challenges will create further opportunities to adop Pravega by new customers and use-cases.

Pravega stores data events in streams (similar to the“topic” grouping in messaging systems). A stream is a durable, elastic, append-only, unbounded sequence of bytes achieving good performance and consistency. Internally, streams are divided into segments. A segment is a shard or partition of the data within a stream. It is worth mentioning that a stream may have multiple segments open for appending events at a given time, which can increase throughput.

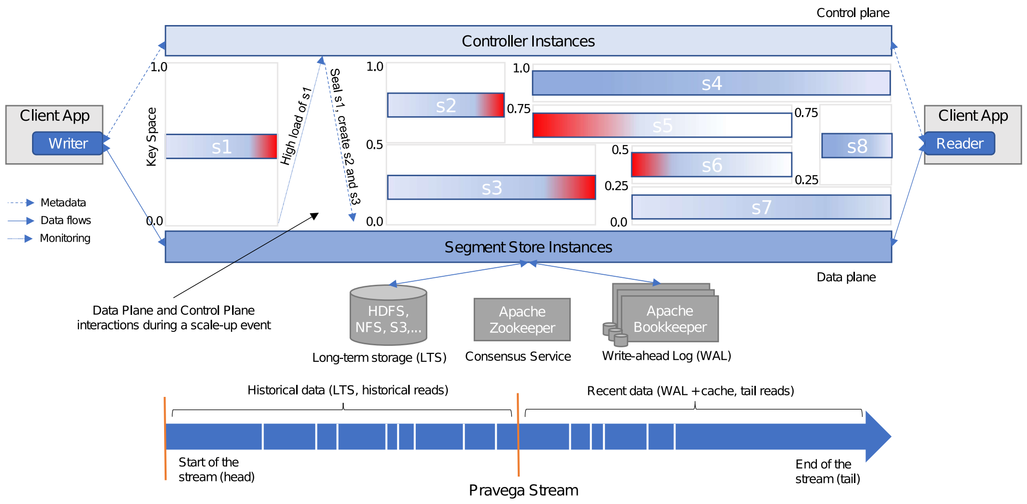


Figure 8: High-level view of the Pravega architecture.

Figure 11 depicts the architecture of Pravega. First, Pravega offers client libraries implementing the server APIs, such as writers and readers, which interact with Pravega server instances either within the same cluster or externally. On the server side, we find the Pravega control plane formed by controller instances. The control plane is primarily responsible for orchestrating all stream life-cycle operations, like creating, updating, scaling, and deleting streams.

Pravega streams are policy-driven and currently the system offers 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). For the latter case, Figure 11 also shows how Pravega can also “split” and “merge” segments dynamically based on the scaling policy defined and the current workload.

The data plane in Pravega handles data requests from clients and is formed by segment store instances. The data plane distributes the segment-related load based on segment containers. More precisely, segment containers 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 (implemented via Apache `Bookkeeper` [21, 22]) is to guarantee that writes are durable with low latency for recovery purposes. 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. To be precise, Pravega achieves both goals by storing historical stream data in a horizontally-scalable storage service.

- Finally, Pravega uses a consensus service (Apache `Zookeeper` [23, 24]) for leader election and general cluster management purposes.

Pravega is open source and it is a sandbox stage project in Cloud Native Computing Foundation (CNCF)[2].

3.2 Envisioned Integration and Challenges

It is widely known that a layered design promotes loose coupling and helps to manage component interactions more effectively, but for proper functioning such layers require interfacing and proper interaction. Here we provide a first specification of such interaction, prior to any API specification, on how the layers will integrate with each other. API specifications are expected to come in the next deliverables.

It is important to note that although month 6 of the project is too early to make big-bet decisions, as the software stack is under ongoing development, some decisions can already be made to narrow down the integration to the most essential touch points.

As a first concrete example, the orchestration and control functionality for the C-Cells is planned to be abstracted by a component called Planner, which will receive the orchestration instructions from layer L3. The Planner will provide the needed interface for the orchestration layer to tell, for instance, that the current execution of a C-Cell, or a collection of C-Cells, must be interrupted and migrated to another site. Similarly, GEDS will provide several storage access abstractions to interact with the C-Cells such as a file system POSIX-like API. Because the core of GEDS is written in C++, and exported as a shared library, an interesting area to work on in the future deliverables will be on how to facilitate the use of GEDS within the C-Cells.

These issues have started to be discussed and explored by the consortium to facilitate future integrations of the CloudSkin software stack. We will discuss the integrations per layer.

3.2.1 Integration of the Learning Plane

With regards to the Learning Plane and "smart" information provider systems, the involved edge and/or Cloud nodes will have a repository of ML models, and a series of modelling and prediction applications.

Integration with the execution layer. The main bridge component between layers L3 and L2 will be the **Planner**, or equivalently an orchestration system (e.g., NearByONE, Lithops [5], etc.). When some placement and provisioning decision is required, the execution layer will contact the Learning Plane. Simply put, the Planner will connect with the Learning Plane through a gRPC service listening to prediction queries. To illustrate, a prediction query can be something similar to: "*With this application, with this characteristics, with these resources available, with this specific placement: how will the application behave?*", and the response may be something such as "*The QoS will be low*" or "*Execution time will be* > 10 *seconds*".

Also, recommendations could be realized of the type: "*from placement* $\mathcal{P}1$, $\mathcal{P}2$ *and* $\mathcal{P}3$, *which is best for application A?*". For each specific prediction or recommendation query, the service will retrieve the most suitable model from the repository, and use it for the prediction or recommendation.

The prediction services can be deployed as "Learning Applications", either as micro-functions, serverless applications, containerized applications such as Kubernetes or K3s, or even C-Cells. Their specific deployment will depend on how and in which environment we want them deployed. Such Learning Applications will contain the corresponding ML libraries and software suits (e.g., PyTorch, Scikit-Learn, etc.), and an interface to be called (e.g., Flask, gRPC). Further, the Learning Applications will be deployed in a centralized node for starters, with the possibility of being deployed locally in

[2] `https://github.com/pravega/pravega`

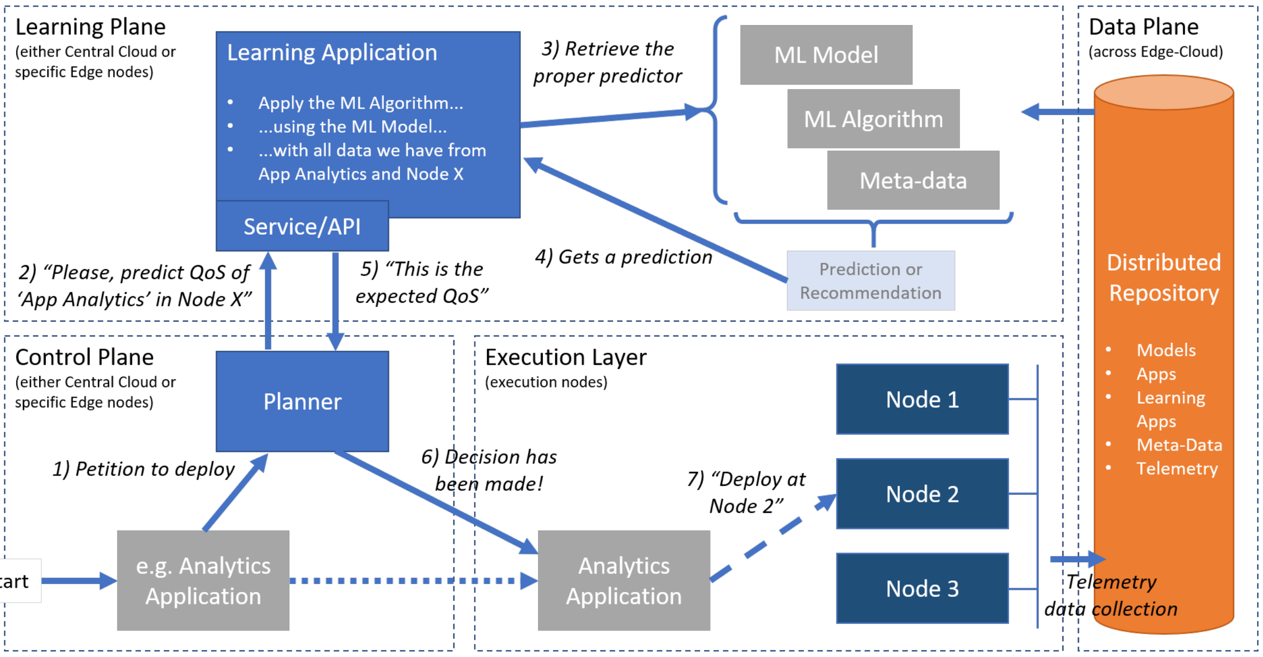

Learning Plane
(either Central Cloud or specific Edge nodes)
Learning Application
Apply the ML Algorithm...
...using the ML Model...
...with all data we have from App Analytics and Node X
Service/API
3) Retrieve the proper predictor
ML Model
ML Algorithm
Meta-data
4) Gets a prediction
Prediction or Recommendation
2) "Please, predict QoS of 'App Analytics' in Node X"
5) "This is the expected QoS"
Data Plane
(across Edge-Cloud)
Distributed Repository
Models
Apps
Learning Apps
Meta-Data
Telemetry
Control Plane
(either Central Cloud or specific Edge nodes)
Planner
1) Petition to deploy
start
e.g. Analytics Application
6) Decision has been made!
Execution Layer
(execution nodes)
Analytics Application
7) "Deploy at Node 2"
Node 1
Node 2
Node 3
Telemetry data collection

Figure 9: A common query for prediction from a Planner (orchestrator) to a Learning model.

edge nodes. Deployment in edge nodes will enable a higher autonomy for near-data nodes, and the easy creation of local models following the principles of *Federated Learning* [25].

The knowledge repository, containing the ML models and any additional metadata for both the learning and decision-making processes needs to be distributed. In a distributed scenario (even a slightly distributed one), the Learning Applications need to retrieve the proper model to predict and recommend. Furthermore, to train the models, the Learning Application will have training and updating functionalities that, given local data, can update or generate new models.

In a federated scenario, each node keeps their models while allowing others nodes to access them for use, or sharing copies through a centralized environment. Also, from differently localized models, the Learning Application can be used to aggregate the models into a more generalized model.

Integration with the infrastructure layer. To support the above functionalities, a means to guarantee fast and reliable access to models and metadata is required. In this regard, GEDS, as an edge-to-core distributed storage system, seems to perfectly fit into the Learning Plane puzzle. Throughout this project, we will explore the benefits of pushing the shared models, the metadata and the encapsulated Learning Applications into GEDS, so that local and remote workers can quickly retrieve a certain Learning Application, a required model, and the required auxiliary metadata.

Canonical integration examples. Here is an example of integrated functioning of the Learning Plane elements against a use case, illustrated in Figure 10. There is a driving circuit with distributed edge nodes along the road, collecting data from the same road and passing vehicles, wile performing local analytics. Besides, we have a centralized HPC cluster, with high computing power and visibility of the edge nodes, but far away of the road and with risk of temporary losing connection to any remote node.

The orchestrator software acting as a Planner is NearByONE, an orchestrator provided by NRB and deployed by Cellnex, who is in charge of the infrastructure. This Planner is in charge to decide in which nodes the analytics application must be deployed. Before doing so, the Planner interacts with the Learning Application in the Cloud, which is periodically receiving information on the status of the edge nodes. The Planner will make a series of requests of the type: "*Where the analytics app will provide a faster response time, given the flow of data from the sensors in the nodes*". As a result, the Learning Application will deliver to the Planner a set of predictions and recommendations on where to deploy the analytics and route data from sensors, enabling the Planner to perform a smart deployment of the application to the nodes.

In another example in Figure 10, we have a vehicle moving along the circuit, and the analytics application in charge of receiving the data from that vehicle must migrate across the infrastructure. The reason for such migration is that we share our infrastructure with other applications, and we do not want to have it deployed at every node, wasting unnecessary resources. The Planner, detecting the trajectory of the vehicle, may in this case ask the Learning Application: "*Which is the next available edge node close enough to my vehicle in which I could deploy the analytics app, without losing much QoS and considering other co-located applications*". The Learning Application may reply with "*Placement $\mathcal{P}1$ will give you QoS $> X$*" + "*Applications there will NOT interfere with your app*". Then, the Planner may decide to move the analytics job to the best placement according to predictions and recommendations.

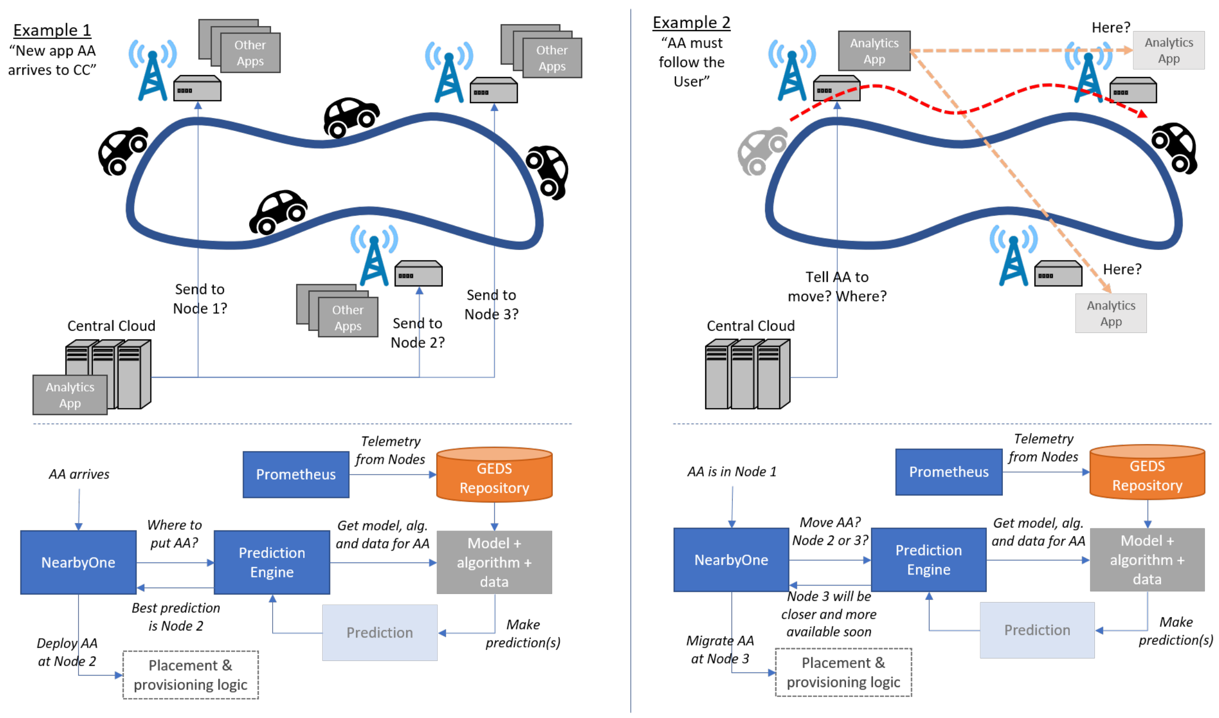


Figure 10: Examples of two cases for Application placement guided by prediction & recommendation on the Mobility Use-Case, with a centralized deployment of the Learning Plane and orchestration.

3.2.2 Integrations at the execution layer

One of the main goals of the execution layer is to support **adaptive virtualization**. That is, depending on the data to process at hand, C-Cells may be found transparently running inside TEEs. There are two ways to transparently execute a C-Cell inside a TEE depending on the granularity of the TEE itself:

- **TEE support at the C-Cell level.** We could integrate with existing WASM runtimes that already support executing WASM modules inside Intel SGX enclaves such as WAMR [26]. The major positive of this strategy is that it would give the ability to spawn as many TEEs as necessary within a single process, meaning that irrespective of whether a C-Cell is co-located inside a TEE or not, the local worker would always have the visibility on which C-Cells are running inside TEEs.
- **TEE support at the runtime level.** Another option would be to integrate with existing "lift-and-shift" programs that transparently run unmodified Linux processes inside TEEs such as SCONE. This means that the whole local C-Cell runtime would run inside a TEE, and hence all

C-Cells scheduled therein would automatically be co-located within the same TEE. In this case, only the Planner with global visibility of the whole execution of C-Cells across the continuum, and not the local workers, will have the visibility to know that the C-Cells running within TEEs.

The first option is theoretically more resource efficient, as it allows to package C-Cells more tightly and reduces memory duplication. However, it requires considerable engineering effort, as all the interactions with the host operating system (e.g., access to the local filesystem, network interface) need to be re-implemented to fit the chosen TEE's `ECall/OCall` mechanism. Moreover, this option also prevents us from leveraging the know-how of another technical partner in the consortium, which is an expert in TEE virtualization through SCONE. Being all things considered, we finally decided to initially explore option two. That is, transparently lift-and-shifting the local C-Cell runtime into a TEE.

Integration with the orchestration layer. One of the important requirements of the C-Cells is the need for **Cloud-edge support**, namely, the concurrent execution in heterogeneous architectures and instruction sets. This means that the **Planner** component needs to be aware of the different supported architectures and incorporate this information into its scheduling policies, taking into consideration the different capabilities of different resources. Lastly, access to shared memory or the network may also be different, yet the interface to C-cells will have to remain the same.

Integration with the infrastructure layer. Integration of the execution layer is not only important as means to share intermediate state between the different workers, but as an efficient way to distribute the WebAssembly modules. One first option would be to leverage a registry such an OCI registry to store the WebAssembly modules[3]. However, GEDS offers a shared object namespace covering the edge and the core. That is, irrespective of its current location, data objects remain always accessible to all computing entities. As a result, we will explore how GEDS can help to efficiently distribute the Wasm binaries across the C-Cells workers.

Finally, it must be noted that GEDS offers efficient data relocation: A computing entity, such as an edge device, may execute a completely ephemeral task, disappearing after task execution, while the tasks data output is relocated within the shared GEDS namespace, to be picked up by a downstream task. This is particularly relevant for C-Cells. Since C-Cells may need to migrate to another locations and be up and running quickly, fast access to the data must be provided to minimize disruptions that may come with migrations.

3.2.3 Integrations at the infrastructure layer

Another important integration point is the one that involves GEDS with Pravega in an effort to benefit the latter. While both systems support tiering in their design, the ultimate goal of tiering is different in each system. Let us signal the differences below:

Once durably written to a write-ahead log (i.e., `Bookkeeper`), events in Pravega are grouped and moved to long-term storage. Clearly, this form of storage is aimed at recording stream data in the long term and serving historical reads. Here is where it gets interesting. In Pravega, long-term storage is an integral part of the write path. This has the negative effect that the system cannot keep ingesting events indefinitely in the case that the long-term storage becomes too slow or unavailable. Concretely, under possible network issues reaching long-term storage, Pravega can only buffer incoming events until the memory cache of Segment Stores is full (e.g., in the order of a few GBs per Segment Store). From this point onward, Pravega would start throttling writers, which may not be acceptable in a use case such as CAS (Computer-Assisted Surgery).

To solve this problem, we can benefit from the tiering properties of GEDS. One of the strengths of GEDS is that can transparently move data across storage tiers, from memory to an external long-term storage system. Thus, if we consider an edge scenario where Pravega ingests data close to the location where it is produced (e.g., surgery room in NCT use-case), GEDS can transparently mediate between Pravega and long-term storage to handle potential network issues. Thanks to the APIs provided by GEDs, it would be feasible to instruct Pravega to store data in GEDS as long-term storage. In turn,

[3]Wam to OCI. `https://github.com/engineerd/wasm-to-oci`

GEDS would be either moving such data directly to the long-term storage system, or temporarily buffering it in local drives in the case of network issues. This integration would greatly help to make the edge side of the CAS surgery use-case (see Fig. 11) tolerant to network failures, which may be common in edge deployments.

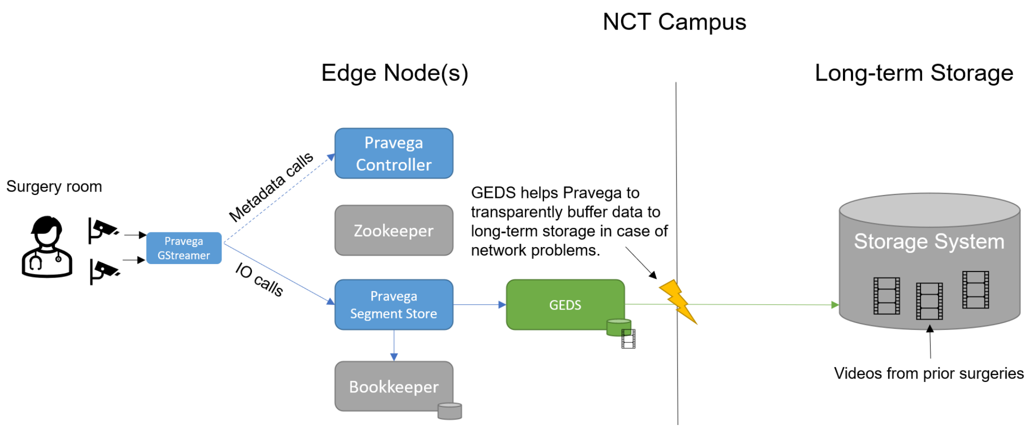


Figure 11: Integration of Pravega and GEDS at the edge.

4 Description of use case scenarios

4.1 Use Case: Orchestration of Applications in Cloud-edge and Mobile Scenarios

Mobile scenarios refer to those in which the infrastructure and workloads are distributed, providing available service to users physically moving from the physically distributed sensors and actuators. For the sake of efficiency, edge computing provides a distributed architecture where applications and data can be off-loaded from the Cloud to be near the users. Examples of such cases are mobile telephone devices, wearables and vehicle-to-everything systems.

4.1.1 Motivation and Challenges

One of the main advantages of moving workloads from the Cloud towards the edge (when possible and feasible) is the capability to set-up, tear-down and migrate applications and services across a distributed set of resources, depending on the geographical need of such application or service. This is the scenario from "mobile" applications that must be deployed following needs and request from users, clients, data and events, instead of being deployed always in a single spot. E.g., in predictive video-analytics (PVA, learning from video and images to produce knowledge from incoming data), it is important to avoid consuming computing power or transmitting unimportant data when there are no events to be analyzed, affecting the application, runtime system and sensors that must be constantly on-line.

Furthermore, on multi-tenant scenarios, such resources and sensor availability must be shared across clients and users, so the application must be deployed specifically on-demand, in-place and on-time. PVA requires certain amount of computing power usually found in the Cloud, so it is rarely brought to the low-power resources in the edge unless adapted to consume the bare minimum, with the corresponding consequences of QoS degradation. To prevent overbooking resources and waste energy, PVA must be deployed in the edge in an intelligent manner (Learning Plane + control plane cooperation) to reduce the limited resources usage.

4.1.2 Data Types for the Use Case

There are two main types of data (datasets), plus additional data to be dealt with (generated by-product data such as models and inferred data), described as follows:

- **Telemetry** from Fog Nodes (i.e., edge and Cloud nodes, including the central node), indicating the computing resources usage (e.g., CPU, memory and bandwidth), quality of service (e.g., response time and latency) and/or energy consumption (e.g. wattage).
- **Multimedia** data coming from the the PVA benchmarking applications in this use case (see Section 5.1), used as large-volume data to be processed on the Fog nodes (i.e., edge, Cloud and intermediate).
- **Models and Inferred** data, generated by the Learning Plane and/or HPDA applications, to be migrated across the Continuum.

Telemetry and multimedia data will be generated in the Fog nodes (mostly in the edge nodes) and processed in-place or transmitted to the central node for processing, depending upon the decisions made by the control plane following the recommendations of the Learning Plane. For this process, a pipeline consisting on data-connectors on data collection, data streaming, data pre-processing, model learning/prediction, and data and model storing will be used.

4.1.3 Data Connectors and Benchmarking

The demonstration tests will consist on deploying applications across the Fog node infrastructure, deployed in Cellnex Fog Computing circuit in Castellolí (see Section 6.1) and using the NearbyONE orchestration engine provided by Nearby Computing, and guided by Barcelona Supercomputing Center's workload placement algorithms and recommenders. Two experiments will be designed and executed for benchmarking:

Experiment 1: Predictive Video Analytics:

A first experiment will focus on deploying ML applications towards learning multi-dimensional time-series (e.g., video data from street cameras), to be deployed on low-power edge devices (e.g., edge nodes in the circuit). Here we will explore the proper PVA application for benchmarking that covers exhaustive resource and time consumption when deployed on the Cloud and capable to be reduced on edge devices, considering as candidate COVA [27]. The tasks of this use case encompass: 1.- Data retrieval, to get global and local video-data for model training and evaluation; 2.- Video pre-processing, to prepare the data for learning and inference; 3.- Model training (in a training and updating stage) video-analytics image detection, 4.- Video-data inference (in a production stage); and 5.- (Meta-)data-storage, to store video-analytics results.

Experiment 2: Mobile Edge Computing:

A second experiment will consist of migrating dynamically the benchmarking application across different nodes, according to the client and infrastructure needs in advance. The tasks to support in this experiment, in addition to those from the first experiment, are: 1.- Model retrieval, to load and migrate models across the infrastructure; and 2.- Model aggregation, to aggregate models generated from different locations.

Such experiments seek to obtain proper application placement in the edge, with proper resources provisioning in an on-line manner in the control plane, while checking how the ML methods from the Learning Plane can provide critical information from predictions. Importantly, the deployment of the benchmarking scenarios will also consider leveraging high-performance storage and trusted environments as part of the data plane, integrating to the global architecture.

4.2 Use Case: Metabolomics

In this use case, EMBL will exploit the CloudSkin technologies to optimize the computing capacities of the METASPACE engine for spatial metabolomics.

4.2.1 Description of the use case

In short, 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 emerging new 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 fast progress in enabling technologies, namely *imaging mass spectrometry* (MS), which have achieved unprecedented sensitivity, coverage, and robustness as they have become accessible to biologists (see Figure 12).

At EMBL, we have developed METASPACE, a global community platform for spatial metabolomics populated by a large community of users [4] (see Figure 13 for further details). In a nutshell, 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.

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

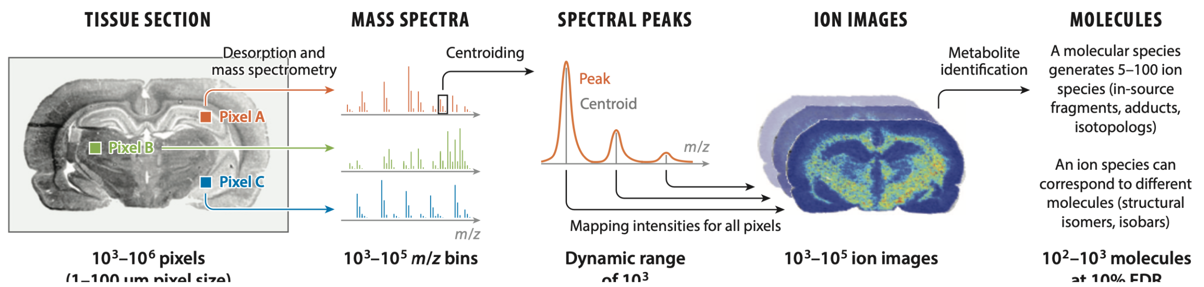
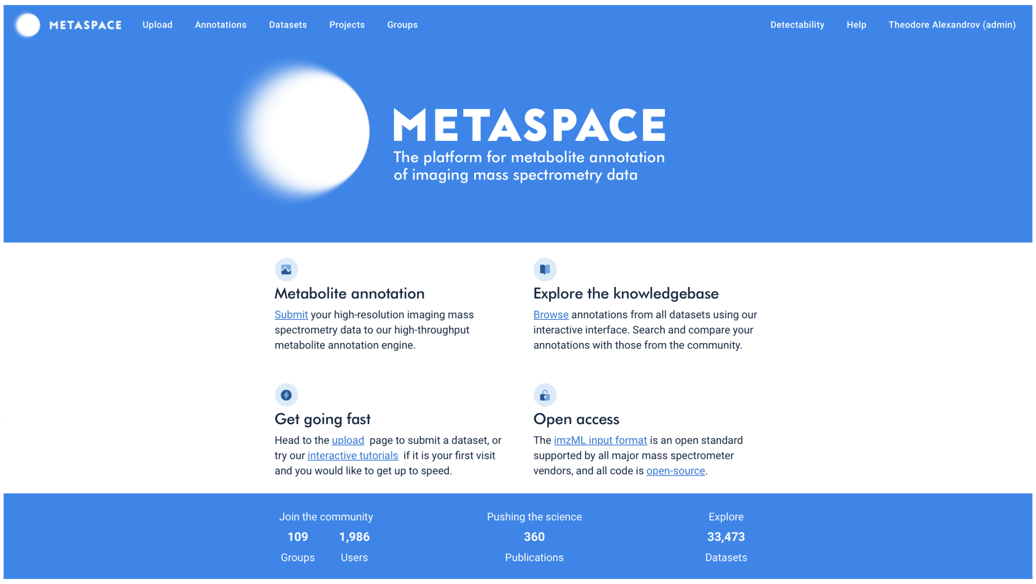

TISSUE SECTION
MASS SPECTRA
SPECTRAL PEAKS
ION IMAGES
MOLECULES
Desorption and mass spectrometry
Pixel A
Pixel B
Pixel C
Centroiding
Peak
Centroid
m/z
Mapping intensities for all pixels
Metabolite identification
A molecular species generates 5–100 ion species (in-source fragments, adducts, isotopologs)
An ion species can correspond to different molecules (structural isomers, isobars)
10^3–10^6 pixels
(1–100 μm pixel size)
10^3–10^5 m/z bins
Dynamic range of 10^3
10^3–10^5 ion images
10^2–10^3 molecules at 10% FDR

Figure 12: 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.



METASPACE

The platform for metabolite annotation of imaging mass spectrometry data

Metabolite annotation

Submit your high-resolution imaging mass spectrometry data to our high-throughput metabolite annotation engine.

Explore the knowledgebase

Browse annotations from all datasets using our interactive interface. Search and compare your annotations with those from the community.

Get going fast

Head to the upload page to submit a dataset, or try our interactive tutorials if it is your first visit and you would like to get up to speed.

Open access

The imzML input format is an open standard supported by all major mass spectrometer vendors, and all code is open-source.

Join the community: 109 Groups, 1,986 Users

Pushing the science: 360 Publications

Explore: 33,473 Datasets

Figure 13: METASPACE landing page showing live statistics and providing access to webapp for data browsing, visualization, and metabolite annotation.

METASPACE was developed earlier in the European Horizon 2020 project METASPACE (2015-2018). METASPACE integrates a high-performance Cloud computing engine, a web interface for data submission, results search, browsing, analysis, and sharing, as well as a knowledgebase of private and public datasets and results from them.

Since 2017, METASPACE became a major tool in spatial metabolomics with over 1,900 registered users from over 100 labs, with many using it every day (see Figure 14). 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 knowledge base of spatial metabolomes.

Importantly, METASPACE requires scalable computing, taking into account the growth of the field (Figure 14) as well as the diversity of the datasets submitted to METASPACE in its nature and size.

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

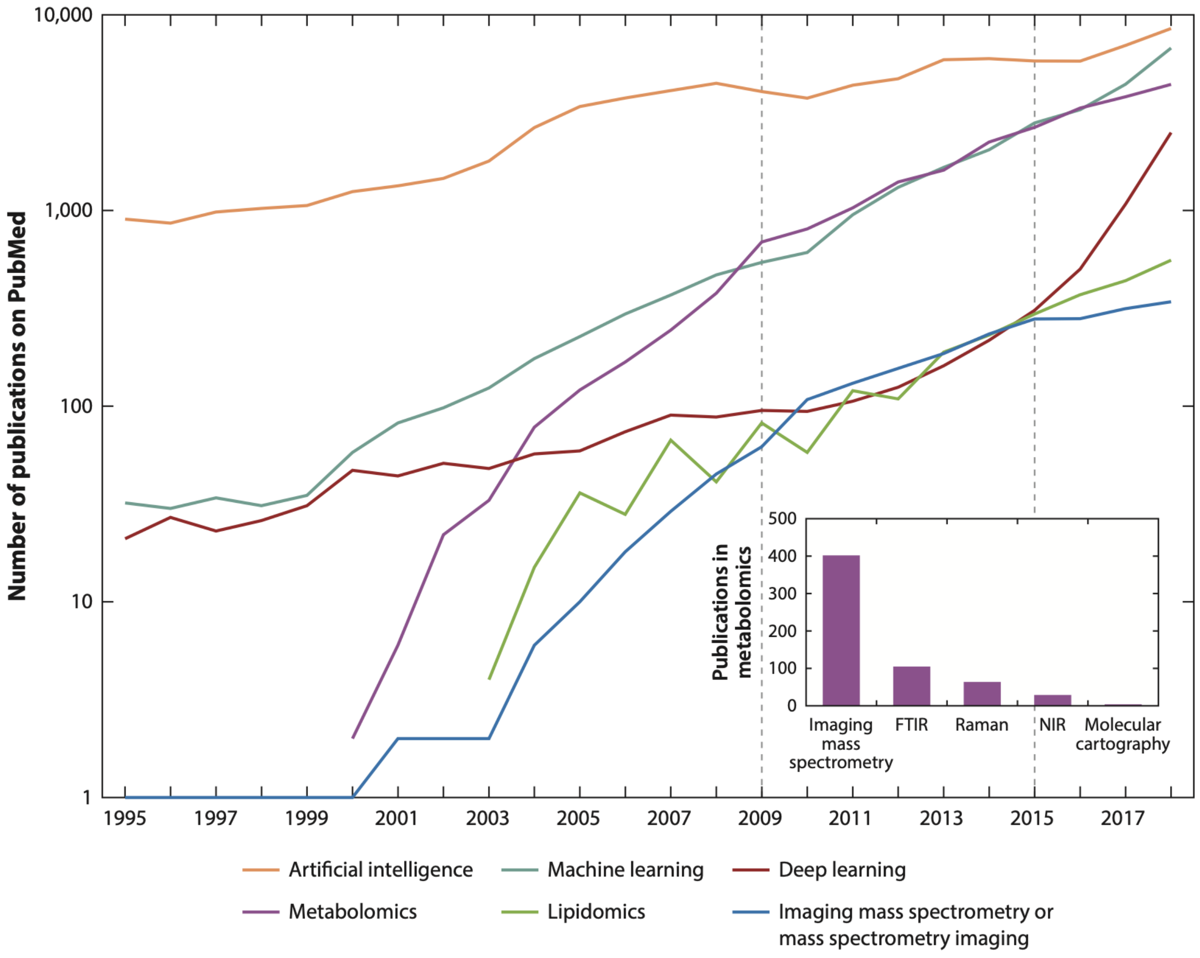


Figure 14: The popularity of different technologies in the life sciences and biomedicine and their evolution over time. The plot shows the numbers of PubMed-indexed publications in a given year containing the keywords shown in the figure key. We highlight three time periods, before 2009, from 2009 until 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.

European project CloudButton (2018-2022) where we have implemented the serverless version of METASPACE using the Lithops framework (see Figure 15). 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.

4.2.2 Datatypes und Datasets

METASPACE stores all datasets in the centroided imzML format `http://imzml.org`. The format can be accessed using Python by using our package pyimzML (`https://github.com/alexandrovteam/pyimzML`) with the documentation available at `https://pyimzml.readthedocs.io/en/latest/`.

4.2.3 Experiments

In particular, EMBL will use the CloudSkin toolkit in the following metabolomics experiments.

Experiment 1: AI-Enabled Optimization of a Machine Learning Service in METASPACE:

METASPACE offers molecular annotation of spatial metabolomics data as a free service to more than 2,000 scientists from over 100 labs worldwide. This includes molecular annotation itself as

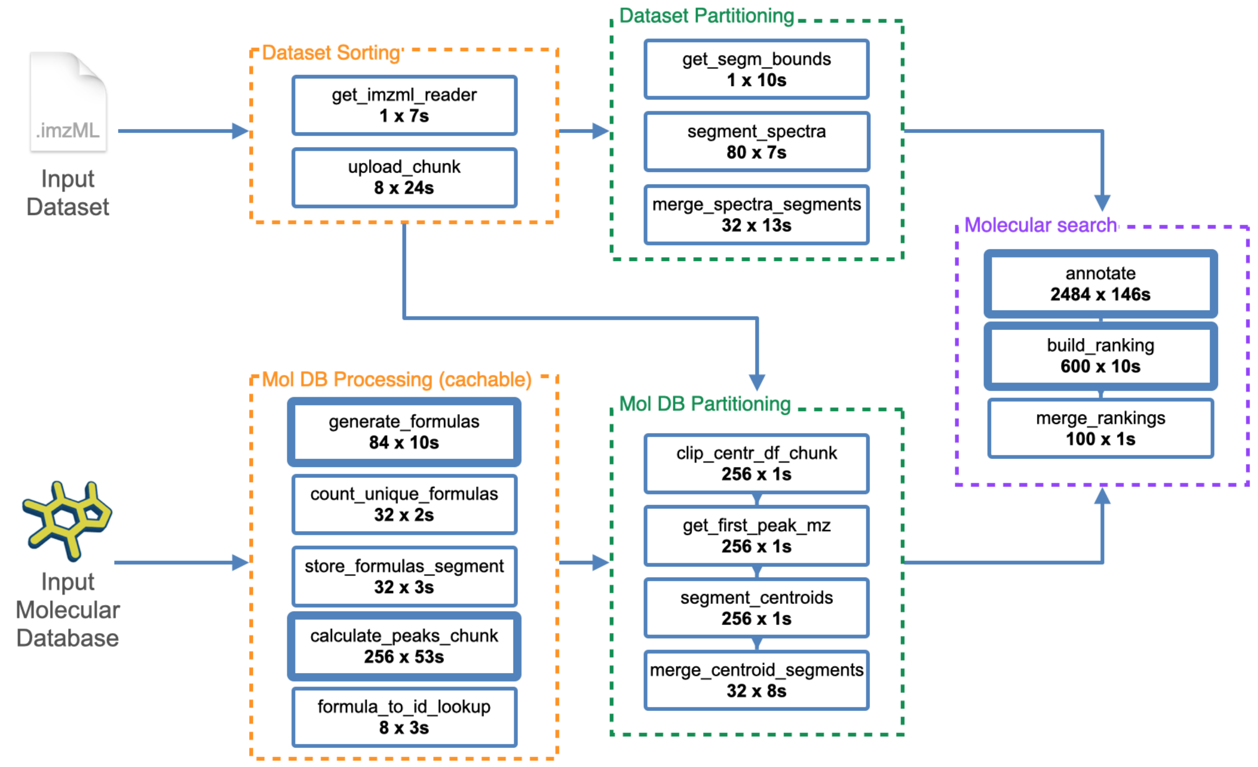

.imzML
Input Dataset
Dataset Sorting
get_imzml_reader 1 x 7s
upload_chunk 8 x 24s
Dataset Partitioning
get_segm_bounds 1 x 10s
segment_spectra 80 x 7s
merge_spectra_segments 32 x 13s
Molecular search
annotate 2484 x 146s
build_ranking 600 x 10s
merge_rankings 100 x 1s
Input Molecular Database
Mol DB Processing (cachable)
generate_formulas 84 x 10s
count_unique_formulas 32 x 2s
store_formulas_segment 32 x 3s
calculate_peaks_chunk 256 x 53s
formula_to_id_lookup 8 x 3s
Mol DB Partitioning
clip_centr_df_chunk 256 x 1s
get_first_peak_mz 256 x 1s
segment_centroids 256 x 1s
merge_centroid_segments 32 x 8s

Figure 15: The architecture of serverless METASPACE.

well as a range of other services. A key additional service is the recognition of off-sample ion images, which helps clean up the data and is applied to all ion images produced by the molecular annotation. This service is performed by invoking a machine learning model that we trained and published earlier (Palmer et. al, Nature Methods 2017 [28]).

In its current implementation, this service is one of the bottlenecks of METASPACE, and the aim of Experiment 1 is to improve it. Currently, the service is implemented as a web service that receives an ion image and returns the classification results (off-sample vs on-sample), with the probability of the classification decision. The model is a PyTorch deep learning model built upon ResNet, and it is deployed and invoked using the Elastic Container Service (ECS) on AWS. The classification takes approximately 1 second per ion image. However, this deployment method is very inefficient because the results of molecular annotation per dataset produce 100-10, 000 ion images, which then need to be submitted to this service and classified one by one. This process can take over 10 minutes and exceed the processing time for one dataset. Considering that METASPACE can receive 100 datasets submitted at once for batch processing, this creates a long queue in the off-sample service. Currently, we address this by increasing the number of ECS containers under high load (up to 4 containers) to process ion images from one dataset in parallel. However, the scaling up and down is slow (+1 / −1) container per minute), leading to suboptimal scalability, runtime, and costs.

In Experiment 1, we will address this issue in two steps:

- First, we will re-implement the off-sample service using a scalable architecture and the Lithops serverless computing framework. To achieve this, we will convert the current machine learning model, trained with Fast.AI and based on PyTorch, into the modern PyTorch format. Then, we will implement the serverless execution of the model using AWS Lambda through Lithops [5]. This approach should provide us with practically unlimited scalability and eliminate the need to constantly run a container in ECS. As a result, we expect a reduction in runtime and overall costs.

- Second, considering that the workload of the off-sample service experiences high burstiness, spiking after dataset annotation and sitting idle for most of the time, we will develop an AI-enabled optimization strategy for predictive management of computing resources. Concretely, this strategy will include determining the number of Lithops workers, memory and computing capacity of exploited AWS Lambda, sizes of batches of ion images to be processed per Lambda, etc. To accomplish this, we will utilize the CloudSkin framework and the expertise of our project partners (BSC and URV).

Experiment 2: AI-Enabled Optimization of Molecular Annotation in METASPACE:

In Experiment 1, we focused on using serverless runtimes and AI/ML to optimize the workload and resources for an additional service in METASPACE, the off-sample ion image recognition. In Experiment 2, however, we will address a much larger problem, which is optimizing the metabolite annotation itself. Metabolite annotation is a challenging task. In the past EU project CloudButton, we re-implemented the METASPACE metabolite annotation, moving it from Apache Spark to a hybrid implementatin that utilizes both a large virtual machine (VM) and the Lithops serverless framework. Such a hybrid VM-Lithops implementation has been used in production since March 2022 and has allowed us to increase scalability. However, there is still a bottleneck in one step (calculated on a VM) that prevents us from processing datasets larger than 250 GB in size. This was not a problem until recently. However, with the introduction of new fast and high-spatial-resolution mass spectrometers, the dataset sizes have increased, and in 2023, we started receiving an increasing number of datasets larger than 250 GB submitted to METASPACE.

In Experiment 2, we will address this issue in two steps:

- First, we will remove the bottleneck in processing large datasets in METASPACE. The current bottleneck is due to the need to sort all mass-to-charge values in a spatial metabolomics dataset. For most datasets, the number of values ranges from 10 million to 1 billion. Unfortunately, for extremely large datasets, it can exceed 1 billion. The current implementation uses a VM with 512 GB of memory, which is expensive and still presents a bottleneck for processing datasets larger than 250 GB. We will address this by leveraging SEER [29], a recent serverless shuffle manager for data analytics, developed by URV (the project coordinator).
- Second, Since the workload of the metabolite annotation service has high burstiness, spiking after dataset annotation and sitting idle for most of the time, we will develop an AI-enabled optimization strategy for predictive management of computing resources. This strategy will involve determining the number of Lithops workers, mostly the memory of underlying AWS Lambda functions, sizes of the batches of ion images to be processed per AWS Lambda, and other relevant factors. To achieve this, we will exploit the CloudSkin framework along with the expertise of our project partners.

4.2.4 Datatypes

We will analyze the spatial metabolomics datasets from METASPACE, including new and historic data. Specifically, a spatial metabolomics dataset represents a hyperspectral image with x–y spatial dimensions and a spectrum (a vector of m/z values each associated with an intensity) stored for each x–y pixel. The data is stored in the `imzML` format and contains a binary part (`.ibd`) and the metadata part (`.imzML`). Moreover, the data on METASPACE is supplied with additional METASPACE-specific minimal metadata in JSON.

4.3 Use Case: Computer-Assisted Surgery

Advancements in technology and digital developments have resulted in valuable data that can improve the surgical outcome. However, the challenge for physicians is the overwhelming amount of data sources available, especially during surgery. The objective of computer-assisted surgery (CAS) is to leverage available data and convert it into valuable information that can assist surgeons throughout surgical procedures.

4.3.1 Challenge in data-driven CAS

Laparoscopic surgery is a minimally invasive surgical technique that is increasingly used to remove various types of tumors in the body. This approach involves making small incisions in the skin and inserting a thin, flexible tube with a camera and specialized surgical tools attached to it. The camera generates real-time images of the surgical site, which can be viewed on a monitor in the operating room. Loss of depth perception and difficult hand-eye coordination are among the most challenging aspects of laparoscopic surgery for surgeons.

CAS can be used to enhance laparoscopic surgery and solve the challenge by providing real-time guidance and precision. It can provide the right assistance functions, such as predicting surgical complications or providing the position of a tumour, by analysing the surgical workflow using video streams from the endoscopic camera. Another key challenge for the realization of computer-assisted laparoscopic surgery is analysing (stereo) video data in real-time with low latency, as computation power needs vary depending on surgery type and progress. Surgical datasets, which can include medical images/videos and patient records, are a critical component boht in the development and implementation of CAS. The sharing and use of multi-centric surgical data for machine-learning due to the technological, ethical, and legal concerns regarding patient and staff privacy, data governance, anonymisation, access control, and secure data exchange presents a bottleneck for CAS.

4.3.2 Surgical tasks

To address this, we will employ the CloudSkin software stack for surgical tasks such as detecting and segmenting organs and tools, determining the current surgical workflow phase, and other processes relating to laparoscopic liver navigation such as disparity estimation. Two central work themes are identified:

- Optimal computational resource allocation across a distributed computation environment (such as a computer cluster), in the context of the varying demands for computation based on the type and phase of surgeries. This is in conjunction with BSC.
- The establishment of a distributed computing solution based on GStreamer & Pravega, with a focus on running several pipelines simultaneously and respectively computational resource allocation. This distributed solution will support two modes: real-time video analytics, local to the respective operating rooms, and large scale computation on data referred to long-term storage, as illustrated in Figure 16. This is in conjunction with DELL.

To validate the achievement of outcomes, different algorithms will be employed. The algorithms and systems will be split into independently-operating algorithms and more complex interacting systems of algorithms, with surgical phase recognition and instrument segmentation as the first candidates for implementation. Once these algorithms have been successfully implemented, the more complex surgical navigation context will be addressed. This involves orchestrating several independent algorithms to illustrate in Augmented Reality (AR) the position of target structures for the surgical team. This will likely involve component algorithms, such as liver segmentation and disparity estimation, deployed on the cloudskin infrastructure, being used within the navigation context.

4.3.3 Datatypes

The primary focus of our project is analyzing video data gathered from operating rooms, which is considered the most detailed and informative form of data. We plan to start by using our in-house data to periodically test and adjust our systems. However, our ultimate goal is to work with the Surgical Facility on the Uniklinikum Campus to create datasets that showcase our project's use case. These datasets will consist of a range of surgical workflows, enabling us to evaluate our infrastructure's ability to handle the corresponding computational demands.

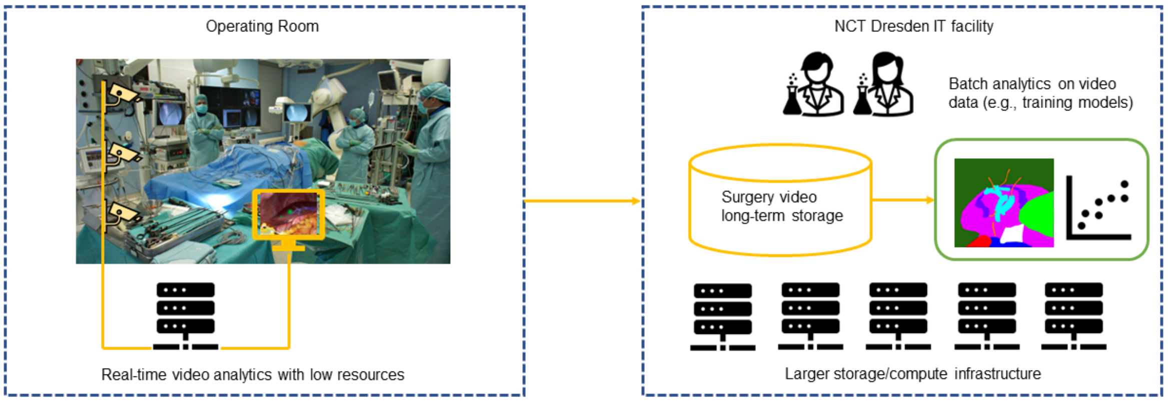

Operating Room
Real-time video analytics with low resources
NCT Dresden IT facility
Batch analytics on video data (e.g., training models)
Surgery video long-term storage
Larger storage/compute infrastructure

Figure 16: Illustration of distributed computing solution built on Pravega. In or adjacent to the Operating Room are limited computing resources which are used to obtain real-time analyses on the data obtained. Separately, this data is committed to long-term storage in a larger IT facility, where, using more substantial compute resources, broader analyses can be performed on a considerable amount of data in addition to machine learning routines.

4.4 Use Case: Agriculture IoT & GeoSpatial Data

4.4.1 Use Case Definition

We establish as the major objective of the use case, the design of a **dataspace** that takes advantage of the information from agricultural and atmospheric sensors. In order to accomplish this goal, we are developing software that allows the negotiation and transfer of information between the different interested stakeholders.

Concretely, data providers should be able to define data and access policies for their owned data, but also making such information accessible by other interested agents. Data access policies that are being studied, include access policies related to the use of data by different categorizations such as the final business sector (to avoid competitors access to the data), benefit (with or without non-profit), country or business type (private, governmental, military, scientific, educational).

One of the objectives is to deal with the complexity of accessing information from sensors in real time, including cases related to cloud edge management, including scenarios of real time access and search in historical data blocks, keeping in mind during the design, the persistence of the information.

The use case proposes access to information from a group of around 500 weather sensors, which are currently monitored by an agricultural IoT software platform. Due to the widespread reluctance of the agricultural sector to relinquish the information from its sensors, concerned about preventing its misuse by its competitors, the location map of the sensors is currently in a negotiation phase. And for this reason, the access to sensor information will be delayed by a few months compared to the initial expectations.

4.4.2 Dataspace Use Case

ALT will concentrate its efforts on the study of dataspace problems, focusing on agricultural and meteorological information systems, obtaining as a result of the process a functional model of data space, which faces the different challenges of the information scheme, proposing specific solutions to the identified problems.

The dataspace will provide an integration model with a cloud data service provider, as a practical use case of how third parties can securely access information from the data space, to feed their own decision support systems.

An example of data integration with geospatial data, or geodata, will also be carried out, serving as an example of how advanced satellite information solutions can be integrated with meteorological data obtained from sensors, connecting through this data space for its integration, improving through this use case, the capability for analysis and practical use of information.

In general, geodata is very diverse and is acquired from dispersed sources, defined in a variety of data type formats. Examples of geospatial data can be spatially dispersed points representing weather stations producing weather data continuously over time, or satellite imagery represented as arrays of thousands of cells describing the surface of the Earth. Other examples of geodata are the following:

- **Vectors and attributes:** Descriptive information about a location, either given as points, lines and polygons.
- **Point clouds:** A collection of co-located charted points that can be recontextured as 3D models.
- **Raster and satellite imagery:** High-resolution images of the Earth, taken from above.
- **Census data:** Released census data tied to specific geographic areas, for the study of community trends.
- **Cell phone data:** Calls routed by satellite, based on GPS location coordinates.
- **Drawn images:** This includes CAD images of buildings or other structures, delivering geographic information as well as architectural data.
- **Social media data:** Social media posts that data scientists can study to identify emerging trends.

Typically, geospatial data occupies an extremely large volume of space. To wit, it is estimated that 100 TB of weather-related data is generated daily [30]. Data management and efficient storage and access is still a big issue for this magnitude of volume.

4.4.3 Geospatial Integration

The analysis capacity of the information will be enriched through a mixed data analysis, which will incorporate both the data provided by the data providers through this data space, as well as those obtained through the geospatial data inputs.

Since the variety of geodata inputs is large, and the budget of ALT is small in comparison with the budgets of other participants in the project, only a proof of concept of what CloudSkin technologies can achieve will be delivered. Still under discussion, this may consist of comparing the information available in the data space with complementary geoestimates, e.g., computed over extended regions of irrigated crop fields, in representative large areas of Peninsular Spain. This will be represented as a workflow of tasks, which are expected to be processed with the help of the CloudSkin platform. For instance, one typical task can be the computation of the `NDVI` (Normalized Difference Vegetation Index) given by:

$$\mathtt{NDVI} = \frac{(\mathtt{NIR} - \mathtt{Red})}{(\mathtt{NIR} + \mathtt{Red})},$$

where `Red` and `NIR` stand for the spectral reflectance measurements acquired in the red (visible) and near-infrared regions, respectively. For instance, `NDVI` is useful to assess the vegetation of an area to identify potential risk of drought or wildfire risk.

Finally, an important part of this objective will consist in analyzing the necessary mechanisms for the integration and efficient use of information from heterogeneous data sources.

4.4.4 Stakeholders

Although different user profiles can be considered for the use and management of the information, we distinguish two main stakeholders by their type of interaction with the platform: Data Providers and Data Consumers.

- **Data Providers** are characterized by providing data accessible to interested users from the dataspace. These users guarantee the data validity, define their information and usefulness, and establish the requirements for access to information.

  Data providers may be not only individual users who provide information from their sensors (such as farmers), but also groups and related entities (Irrigation Associations), institutions (Government) or companies that manage the sensor data.

- **Data Consumers** are a user profile who wish to access the data for multiple reasons, both to resell it (brokers), and to use it on their platforms (service companies) or analyze it for other purposes (scientific, governmental).

5 Benchmarking framework

5.1 Use Case: Orchestration of Applications in Cloud-edge and Mobile Scenarios

In order to assess the efficacy and efficiency of the implemented zero-touch automation control loop, we intend to evaluate the information flow within the control and data planes in cutting-edge scenarios involving connected vehicles. These scenarios may involve the deployment of containerized applications compatible with 5G/G6 networks at the network edge. These applications, such as data analytics for object detection or XR-enabled infotainment services, necessitate seamless coordination within the cloud continuum in an optimally efficient manner.

To achieve the orchestration of such applications, the learning plane will analyze the collected data and provide decisions or suggestions to the orchestrator. As part of our benchmark scenarios, we will examine decisions pertaining to service placement or migration within the cloud continuum. These decisions will be based on various metrics of interest, which may encompass factors such as energy availability, latency requirements and service level objectives, among others.

5.2 Use Case: Metabolomics

For benchmarking the experiments in the metabolomics use-case, we will use the following criteria:

1. We will consider the maximal size of the dataset possible for processing. Currently, it is limited by the memory of the VM because of the need for in-memory sorting that limits it to 150 GB. Our requirement for this parameter is to be able to process a dataset as large as 1 TB that is necessary taking into account the increasing spatial resolution and speed of modern mass spectrometers.

2. We will consider the number of datasets that can be processed in a batch mode. For reprocessing all public historic datasets on METASPACE, this should be as large as $8,000$. Nonetheless, we expect that this number will grow at least to $10,000$ during the scope of the CloudSkin project.

3. For the hybrid experiment, we will estimate the processing time, along with the amount of deployment, administration, and support effort needed for processing data on the EMBL HPC cluster compared to AWS. These are the critical parameters that will determine the feasibility and attractiveness of the hybrid processing compared to the currently-used version on AWS.

5.3 Use Case: Computer-Assisted Surgery

As it stands, there already exists a means of performing distributed computing within the NCT, namely Robot Operating System (ROS). Because of this, any distributed computing solution that is developed in the course of this project should be evaluated against ROS for speed and suitability for dealing with multi-operating room scenarios.

Assessment of distributed computing will take place on infrastructure available to the NCT on the campus of the University Hospital in Dresden. The computing resources available vary considerably, and orchestrating their use most appropriately will test the mechanism for the optimal allocation of computing resources.

Within a multi-operating-room scenario, the developed strategy for computer-assisted surgery will be validated using two benchmarking cases: (1) surgical phase and scene recognition; (2) surgical navigation-related tools.

Surgical videos with phase annotation and scene analysis method will be combined to determine the current state of a surgery. We will analyse the video stream of an endoscope live during surgery as part of providing the surgical staff with assistive visualisations. Currently, this is difficult as different machine learning models are active at the same time, depending on the surgical phase and progress. In one phase we might just want to detect an organ in real-time and display to the surgeon where the organ is located via augmented reality in the endoscope view, but in another phase of the surgery, more analysis might be required, for example segmentation of multiple objects while performing a 3D reconstruction and performing biomechanical modelling (all via deep learning). So, the GPU power that is required might change dynamically during surgery, and the video stream can be very large.

In surgical navigation pipelines, a 3D reconstruction of the surgical scene is performed based on stereo video. In one of the steps, relevant organs are detected (e.g., a liver), and registration with preoperative data is performed for the augmented reality of the surgical video. This is a well known very challenging case, as we have several (expensive) algorithms that build upon each other, but have to be very fast as the augmented reality display should not be outdated.

To evaluate streaming and distributed continuum applications, we have to consider different key factors:

- Performance (Frame Rate, Latency and throughput)
- Scalability (possibility to handle larger workloads as needed)
- Reliability (especially to handle failures gracefully)
- Security, and
- User experience

AI-assisted surgery, particularly in the case of involving real-time video processing requires efficient and reliable data management solutions. Video data from these procedures can be extensive and processing this data in real-time is essential for assisting the surgeon and successful operations. As such, a data streaming platform is required. Pravega is an open-source data streaming platform which is an ideal choice for such a use case for several reasons:

1. **Streams for Continuous Data**: AI-assisted surgery can generate continuous streams of video data. Pravega's primary abstraction is a "stream", which has been designed to handle continuous and unbounded sequences of bytes. This makes it suitable for storing and managing real-time video data from surgeries.

2. **Scalability**: Pravega can automatically scale to handle varying loads of data. In the case of AI-assisted surgery, the amount of data can fluctuate widely depending on the complexity of the procedure, the resolution of the video, and many other factors. Pravega's auto-scaling feature ensures that it can efficiently handle these fluctuations.

3. **Consistency and Atomicity**: Pravega supports transactional writes, implying that the data is only visible after the transaction is committed. This helps ensure the consistency and atomicity of the data, critical when dealing with medical procedures where data accuracy is paramount.

4. **Fault Tolerance**: Surgeries cannot afford system failures or data loss. Pravega has robust mechanisms for handling failures and preventing data loss. Even if a part of the system fails, Pravega's architecture ensures that the data remains available and consistent.

5. **Real-Time Processing**: Pravega's architecture supports real-time processing, crucial in an AI-assisted surgery scenario where the video data needs to be processed in real-time to assist surgeons during the operation.

6. **Security**: In healthcare, data privacy and security are paramount. Pravega has built-in security features like encryption and access control to ensure that sensitive data is kept secure.

Pravega's capabilities align well with the requirements of this use case, making it an ideal choice for such scenarios.

5.4 Use Case: Agriculture IoT & GeoSpatial Data

5.4.1 Problems identified

Different characteristics have been identified that must be analyzed, in order to apply solutions that allow the viability of the use case:

1. **Legal.** Compliance with international legislation is a critical aspect for the treatment of data. User data must be reliable, and comply with the rules of use and access defined by the owner and current legislation. For this, we will define a system of contracts and legal restrictions by country.
2. **Data consistency.** The data to be useful must be understandable, and be in measures known by all users. This problem is solved through a data dictionary, which will allow the parties to know the semantics of the data.
3. **Variable data formats.** The heterogeneity of sensor types and data input formats is a challenge for managing new types of data. Although it cannot be avoided that each manufacturer or type of sensor defines its own data communication interface, the application of adaptive solutions using database schemas and the study of solutions at the edge of the cloud, allows us to define proposals for data access standardization.
4. **Data persistence.** It is as important to offer the data as to guarantee that it is not modified. Within the current project, we will study the following scenarios, which in turn present unique complexities for each scenario from a scalability point of view:
   - The first case study assumes that the data provider itself offers direct access to the data, using the dataspace as an access authenticator.
   - The second option, uses the dataspace as a guarantee seal for the data, storing it in its databases or private persistence solutions, and offering access to the consumers without using the provider infrastructure.
   - At last, the data space can use third-party tools, such as Pravega [3], to guarantee seamless access and persistence of information.
5. **User contracts.** User identification and role management is essential to ensure the proper use of information. We will study how to establish, through contracts and role definition mechanisms, the necessary guarantees to ensure secure access to information.

5.4.2 Use case evaluation indicators and methodology

- **Operability tests**. Functional operability tests of the dataspace will be established, by creating users with different profiles (data producers, data consumers, etc.), and by staging examples of contracting and use of the data.
- **Performance tests**. The impacts on the performance of the platform will be analyzed, taking into account the design decisions made on access to the data. The performance tests will include the execution of a geospatial workflow implemented with the technologies of the project as means of verification.
- **Output emulation**. A sensor data reception system will be made available to the project with a pre-established periodicity, which allows partners to carry out analysis and integration of the data with their projects, using this use case with real data.

6 Testbed Specifications

6.1 Use Case: Orchestration of Applications in Cloud-edge and Mobile Scenarios

The use case will be hosted at the Castellolí trial site, which will provide all the necessary infrastructure. Castellolí trial site is located just 60 km away from Barcelona, very close to the Natural Parc of Montserrat. Given this location, the track presents a special geography: climbs, slopes, bridges, curves and hills. The Circuit ParcMotor (located in the Castellolí trial site) covers an area of 100 hectares offering top-class facilities for every discipline and level of motorsport. This circuit, represented in Figure 17, has been equipped with C-V2X, 5G and edge computing technologies, wireless network, with coverage throughout the venue. The track has high-definition cameras for tracking vehicles on the track and on-board units in the vehicles themselves for the transmission of telemetry, video and voice data. The IoT network, based on LPWA Sigfox technology, is deployed on the circuit with data management and analysis capabilities.

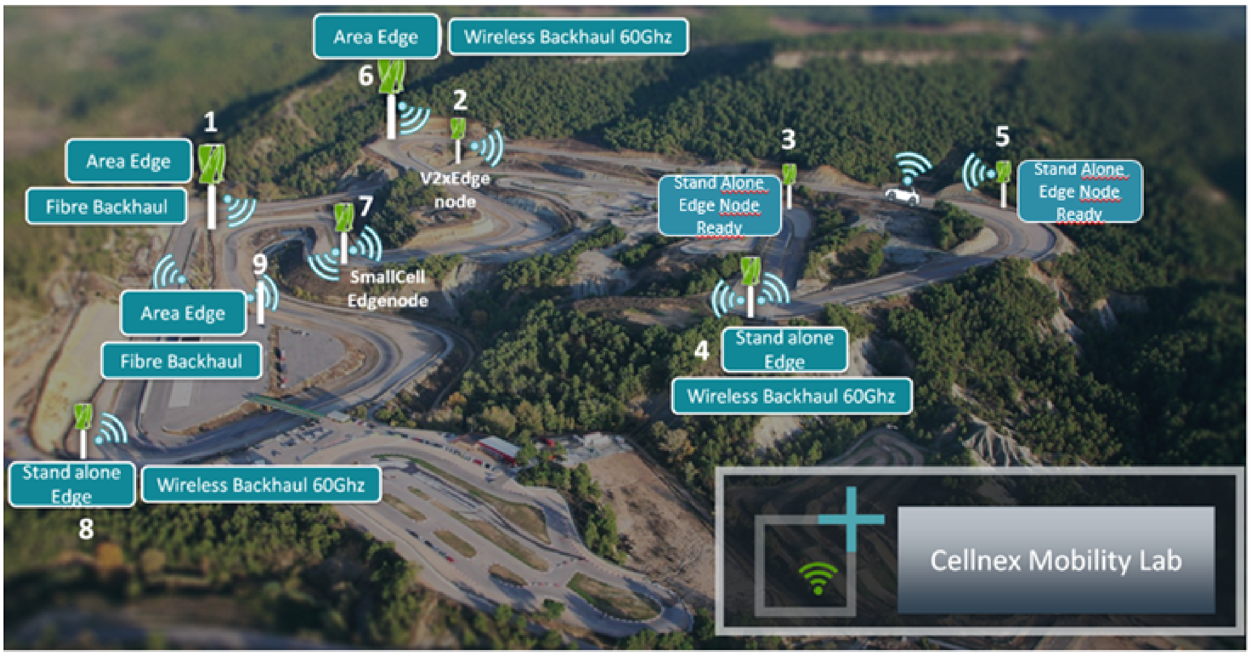


Figure 17: Node distribution of the ParcMotor Circuit 4.1 as an example.

For the use case to be developed, the Castellolí platform will offer two types of nodes, the Green nodes and a Grid Site. The Green nodes are self-sustainable, all the consumed energy is produced by its solar panels. Specific algorithms will allow to manage the energy and display the consumes of each device connected to the node (cameras, RU, servers etc). Data about the, indoor and outdoor, humidity and temperature in each node will also be displayed and manageable. We can predict the environmental variables or know the actual atmospheric conditions due to the IA created in base of the sensor network. In case the weather conditions are not good enough to sustain the energetic cost of a green node, the main services will be transferred into the Grid Site until the weather can handle its energetic cost. Figure 18 shows an example of a Green node indicating all the instrumentation installed in it. The Grid Site is connected into the electricity distribution system and also to fiber. This allows to assure the service continuity and run the services that may consume higher levels of consumption in the systems of this node. In the Control Room, two Lenovo ThinkSystem SR650 servers will be installed which are designed to handle a wide range of workloads, such as databases, virtualization and cloud computing, virtual desktop infrastructure (VDI), enterprise applications, collaboration/email, and business analytics and big data. In the different nodes there will be installed the Lenovo ThinkSystem SE350 edge nodes which will hold the different services, composed of Dockers and containers, as indicated in Figure 18.

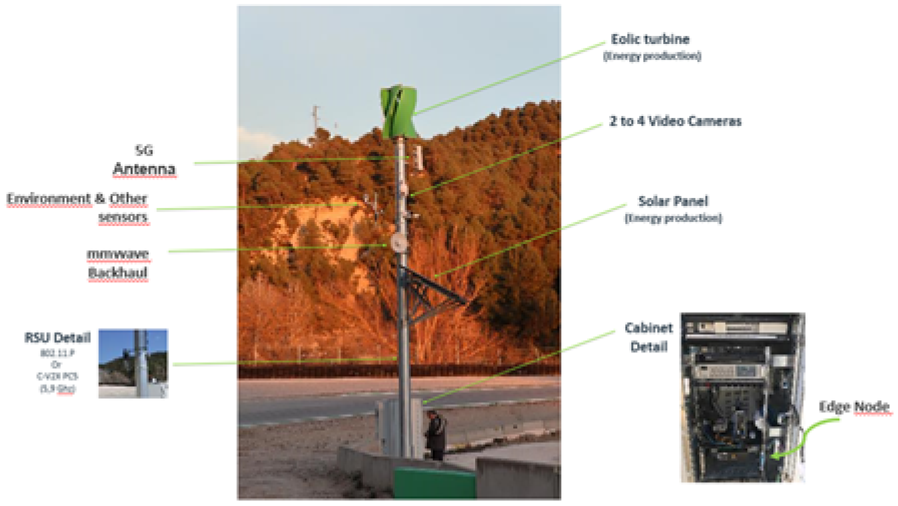


Figure 18: Green Site characteristics.

6.2 Use Case: Metabolomics

We will use AWS, the main cloud where METASPACE is deployed. For the hybrid experiment, we will use the cluster at EMBL Heidelberg. This is a highly resilient dual-data centre infrastructure built upon state-of-the-art technology and, according to demand, is flexibly scalable in terms of capacity and performance. Based upon the latest Intel and AMD technology, this cluster currently provides access to more than 14,000 CPU cores and a total RAM/memory of 90 TB.

The cluster also integrates compute power using GPU hardware, based on recent Nvidia Pascal hardware and adding about 330 TFLOPS of floating point performance. The underlying ultra-fast parallel file system is designed to support the massive I/O needs commonly found in data-intensive bioinformatics and computational biology HPC workloads.

6.3 Use Case: Computer-Assisted Surgery

Testing the NCT use case simply requires a distributed network of computers, which exists within the University Hospital campus in Dresden. The computing resources available consist of a range of computers with varying CPUs and GPUs, which allows for the optimal resource allocation to be tested in the distributed computing context.

AI-Assisted Surgery System using Pravega

Phase 1: Single VM Prototype
Objective: Establish a functioning prototype of the system on a single VM, running Pravega and AI models for video processing. This initial phase simplifies integration and testing.
We will accomplish the following:

- Set up a fully configured VM with a GPU, running necessary software including Pravega, the AI frameworks, and other necessary tools.
- Develop a program to generate or capture surgery video data and feed it into Pravega.
- Develop or integrate an AI model for video processing, optimized for GPU execution.
- Integrate the video stream and AI model with the data stream from Pravega.
- Conduct performance tests to demonstrate real-time video data processing.

Phase 2: Kubernetes Cluster Deployment
Objective: Scale the prototype by deploying it on a Kubernetes (K8s) cluster for distribution and fault tolerance. This is the intended way to deploy and use Pravega
We will accomplish the following:

- Set up a fully configured K8s cluster with necessary hardware and GPU resources.
- Create Docker images of the AI application for deployment on K8s.
- Successfully deploy Pravega and the AI application pods on the K8s cluster.
- Demonstrate the system's scalability by adjusting the number of AI processing pods based on the load.
- Conduct tests to show the system's ability to handle larger loads and recover from failures.

Phase 3: Hospital Deployment & Real-Time Operation
Objective: Implement the system in the hospital environment and validate its performance with real-time surgery video data.
We will accomplish the following:

- Set up a ready-to-use K8s cluster which can be securely connected to the an NCT centre which achieves the required minimum latency.
- Successfully deploy the system at NCT, connected to the surgical video equipment.
- Conduct real-time testing under simulated local surgery conditions.
- Evaluate the AI-assisted surgery system, including technical performance and effectiveness in assisting surgeries.
- Prepare a post-deployment improvement plan based on evaluation results.

A diagram describing the deployment phases can be seen in Figure 19.

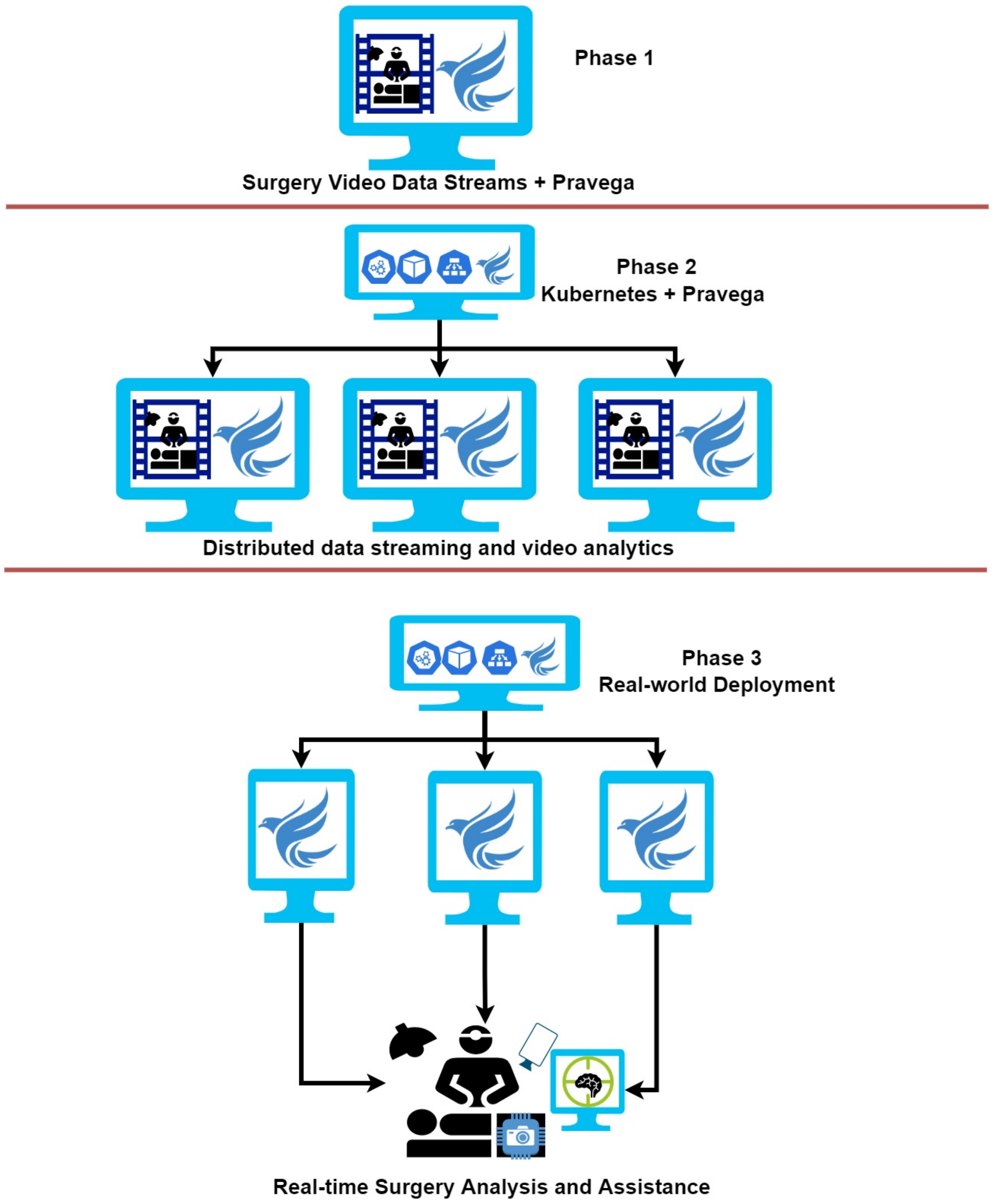

Phase 1
Surgery Video Data Streams + Pravega
Phase 2
Kubernetes + Pravega
Distributed data streaming and video analytics
Phase 3
Real-world Deployment
Real-time Surgery Analysis and Assistance

Figure 19: Diagram showing deployment phases of Pravega with NCT use case.

6.4 Use Case: Agriculture IoT & GeoSpatial Data

6.4.1 Development approach

The project starts taking a series of decisions about how to build the dataspace:

- A user (data provider) will be able to register what we have called datasets. These datasets are a set of data with a homogeneous format, which have been generated by one or more sensors. To each dataset we associate an engine. This functions as a data parsing engine, and has been defined in this way due to the possibility of using the RESTful API in most cases. In this use case, each input dataset will be transformed and load it into the database. Since each dataset will have a different data format, it has been designed so that after receiving data, e.g., in JSON format, the engine will serialize data to store the information in a data blob, which is much more efficient in disk terms than save information, e.g., as textual JSON documents.

Here we will study the use of C-Cells to carry out format transformations on the edge side.

- Each individual data entry in that dataset is a dataread. It is important to note that the datareads can be geo-positioned individually, since we may have obtained said information from different sources. This approach is the opposite of the design of managing the individual registration of sensors, forcing them to be geo-positioned and adding a sensor id to each dataread. This design decision has been made after identifying it as a more flexible method than datareads that can arrive from ephemeral sources. This decision significantly speeds up the start of sending data to the system by the sensors, since it is not necessary to carry out a previous definition and registration of each sensor.

6.4.2 Description of the databases

The substrate of the dataspace is based on an initial database schema that will manage the following information.

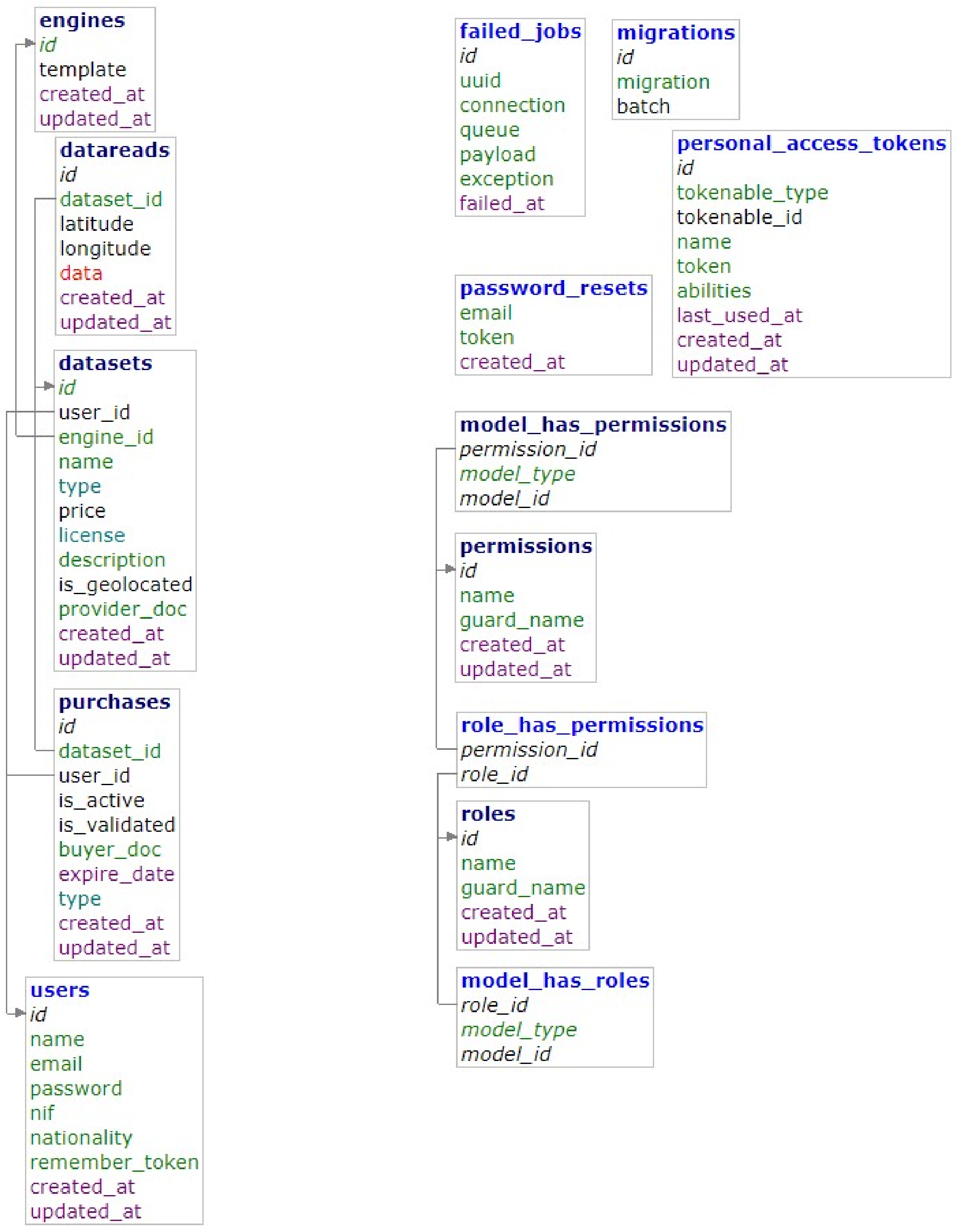


Figure 20: Database structure for the Agriculture dataspace.

On the one hand, and on the right side of the image, we illustrate the information tables related to the user login, the defined roles (three initially defined), the password recovery system, and the necessary information for application management. On the other hand, we define on the left side of the image, the tables that will record the knowledge model of the dataspace.

Next, we describe the use of main data tables:

- **Engines:** In addition to its `id` and its `created-at`, `updated-at` (fields that are present in all the tables and that are internal management fields), the table defines a template. That template is a JSON that indicates how to serialize the data. This is where you would indicate, for example, that "`temperature`" is a 32-bit float with Celsius as the unit. This design decision facilitates serialization to disk (which greatly optimizes storage).

- **Datasets:** Reference to `engine-id`, to `user-id` (user to which the dataset belongs), has a `type` field (rental, purchase, type of use of the information), has a `price` (rental or sale), a `license` (that we have initially defined to reflect the regulations for legal use of the information, though we have not yet specified the management model), a `description` of the dataset (which the user can see), the obligation to locate its datareads (i.e., `boolean`, to indicate if it is mandatory to send GPS coordinates in each dataread) and finally, `provider-doc` to upload a document type "data consent" or with the license.
- **Purchases:** Created to allow tracking of which user has purchased/rented or accessed a given dataset. More specifically, there is an `is-active` field especially useful for rentals. When the "subscription" expires this field will be set to `False`. Also, there is an `is-validated` field (to ensure that the contract has been signed and reviewed), documentation signed by the buyer, an `expiration date` (for purchases it is not limited and for rentals it will be the end date of the contract), and a `type` to indicate if it is a purchase or rental or the type of data usage.
- **Users:** User table. At the moment in an initial version, it includes nationality in case the datasets are restricted by nationality or geographic area of the user in the use license (e.g., it can only be used in Spain).

6.5 KIONetworks: Testbed Architecture

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 CloudSkin project.

KIO Networks, as a service provider working on this type of initiatives, provides the company with growth both in knowledge and in integration with the rest of service providers, with the main objective of standardizing and industrializing services, thus building Clouds integrated with load and data movement capacity between them. KIO Networks is highly interested in the exploitation of Cloud technologies involving data management, as we are company that offers Cloud and edge infrastructures to public institutions in Spain.

KIO Networks will provide Cloud infrastructure services on datacentres 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 such as SSDs.

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.

6.5.1 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. VDC provides dedicated computing resources (e.g., 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 (see Figure 21).

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 will 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, 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.

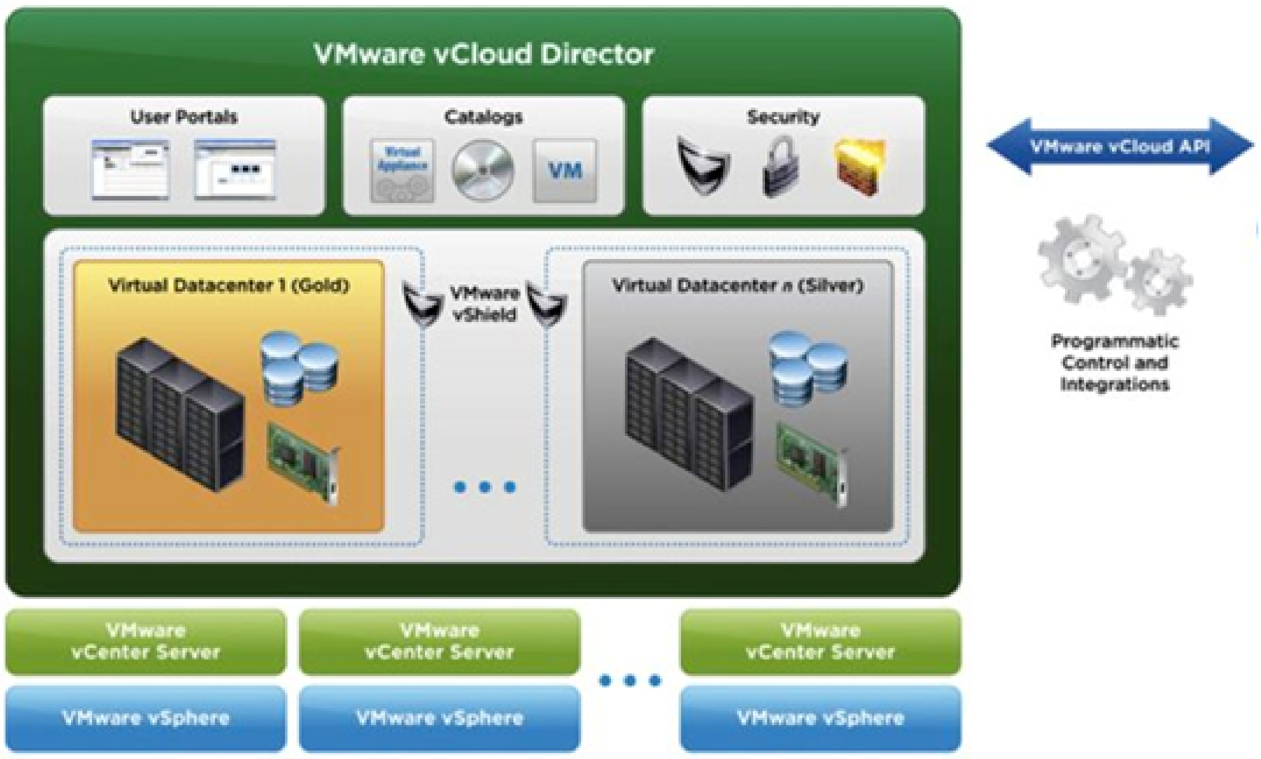
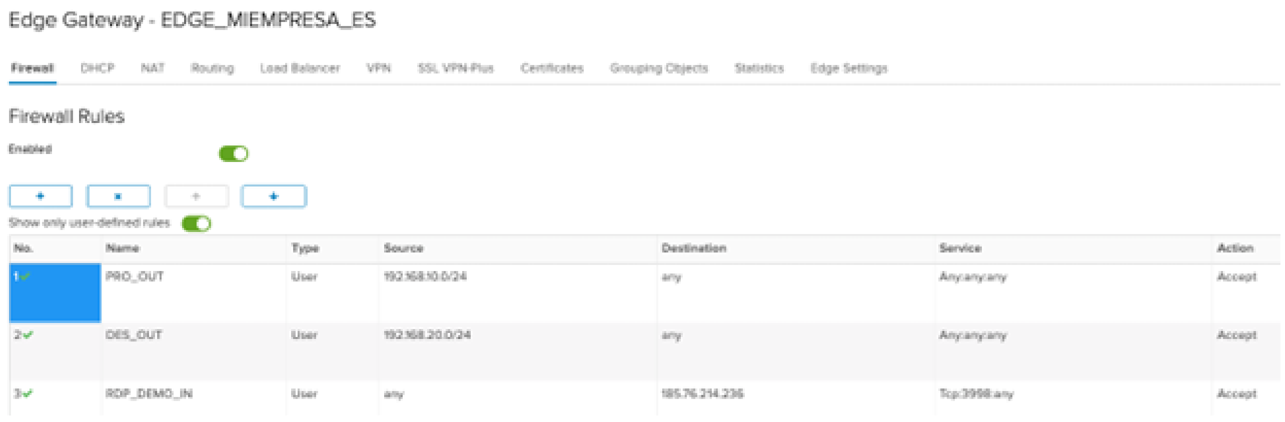


Figure 21: KIO Networks Virtual Data Center (VDC) in Spain.

The VDC platform is delivered from KIO Networks TIER IV certified data centres 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, in order 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% guaranteed availability.

KIO Networks will provide an integrated networking and perimeter security management. That is, the VDC web control enables the administrator to manage virtual networks and perimeter security policies. 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 management, DHCP servers (see Figure 22).


Edge Gateway - EDGE_MIEMPRESA_ES

Firewall Rules

Enabled

Show only user-defined rules

No.	Name	Type	Source	Destination	Service	Action
1✓	PRO_OUT	User	192.168.10.0/24	any	Any:any:any	Accept
2✓	DES_OUT	User	192.168.20.0/24	any	Any:any:any	Accept
3✓	RDP_DEMO_IN	User	any	185.76.214.236	Tcp:3998:any	Accept

Figure 22: Integrated networking and perimeter security management on the edge side.

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.

6.5.2 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 data-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.

	At KIO, we use the latest technology in this new field. We have Ampere cards from the NVIDIA enterprise range.
	Specifically, the A100 80GB, which allows us to market a series of computing capacity options in a multi-tenant format.
	This type of technology is widely used in statistical inference, ML, mathematical analysis, AI or deep learning (DL).
	This service is delivered as part of our Go1Kloud service, which allows customers to unify traditional servers, SAP Business One, and now GPU-based loads.

Table 2: Features of the KIO Networks GPU-based testbed.

6.5.3 KIO Networks Cloud Computing S3 Architecture (S3)

This is the network architecture of KIO Networks Clients to provide access to AWS S3 storage service (see Figure 23).

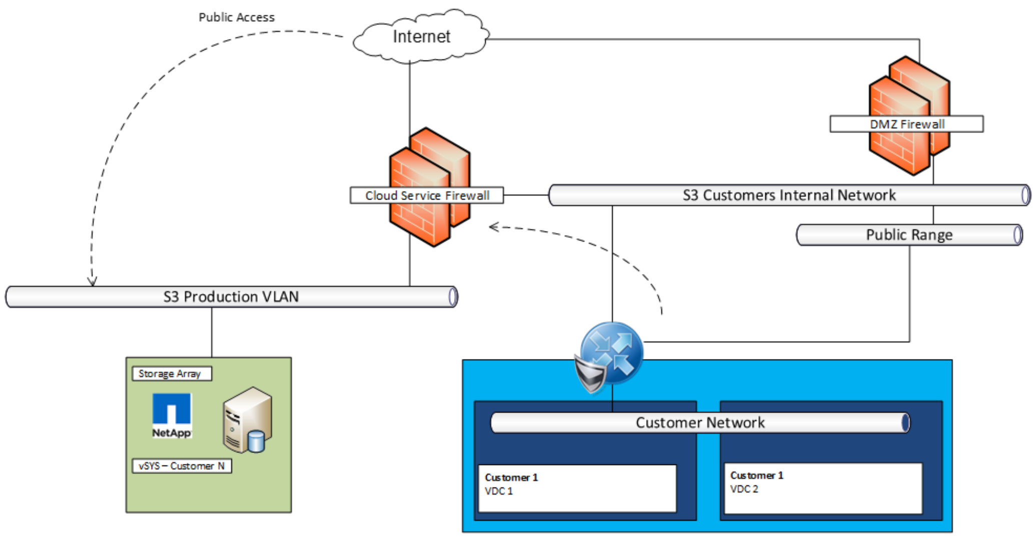


Figure 23: Available S3 public service in KIO Networks.

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 like Financial Services, Online Transactional Processing (OLTP), High-Performance Computing (HPC), Data lakes, Cloud native applications, and Mobile applications.
- **Cold Data** (SANKLOUDBCK): This type of data will be used to backup data on a SATA disks to guarantee a coherent backup for non-critical files or sporadic files.

6.5.4 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 (see Figure 24).

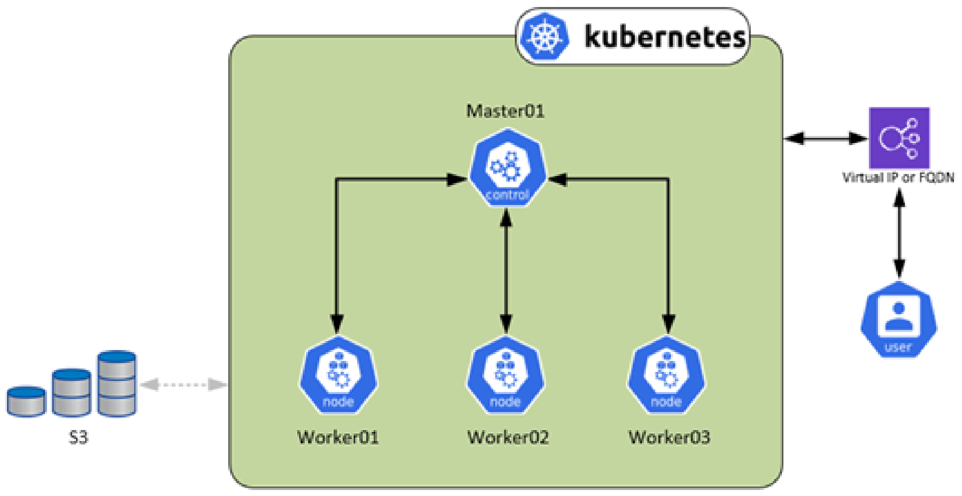


Figure 24: Typical Kubernetes deployment at KIO datacenters.

The Kubernetes environment proposed for the project will be based on an architecture with the characteristics that can be seen in the previous figure. 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.

7 Conclusions

This deliverable presents the initial specifications of the global CloudSkin platform 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 fully comply with all the objectives and main innovations of the project. More concretely, it is intended to ensure that our novel CloudSkin platform meets the expected expectations in terms of designing a cognitive cloud continuum platform to fully exploit the available Cloud-edge heterogeneous resources with a unified and secure execution abstraction, finding the "sweet spot" between the cloud and the edge, and smartly adapting to changes in application behavior via AI. The CloudSkin platform will design a high-performance infrastructure for the cloud continuum, tailored to the short-lived, also bursty, execution of Cloud-edge tasks.

In this deliverable, we have been able to state to a large extent the following information:

- The components forming the global CloudSkin layered architecture have been detailed, as well as the software frameworks used for its implementation. Then, an initial integration of our software stack with the different use cases has been already described.
- For each use case, the first initial requirements, specifications and objectives have been accrued, along with the characteristics of the datasets when applicable. Likewise, the experiments and their related execution workflows have been presented in detail.
- For the evaluation of each use case, the different test environments and experiments have been also introduced in detail. Also, KIO, as a testbed provider, has given the specs of the different environments available at KIO for running some of the use-case experiments.

References

[1] A. Zakai, A. Haas, A. Rossberg, B. Titzer, D. Gohman, D. Schuff, J. Bastien, L. Wagner, and M. Holman, "Bringing the web up to speed with webassembly," in ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), (Barcelona, Madrid), 2017.

[2] The Linux Foundation, "Kubernetes." `https://kubernetes.io/`, 2020.

[3] N. Computing, "Nearbyone edge orchestrator." `https://www.nearbycomputing.com/wp-content/uploads/2021/08/NearbyOne-Product-Data-Sheet-v2.0.pdf`, 2021.

[4] J. Sampé, G. Vernik, M. Sánchez-Artigas, and P. García-López, "Serverless data analytics in the ibm cloud," in 19th ACM/IFIP Middleware Conference Industry (Middleware'18), pp. 1–7, 2018.

[5] J. Sampe, P. Garcia-Lopez, M. Sanchez-Artigas, G. Vernik, P. Roca-Llaberia, and A. Arjona, "Toward multicloud access transparency in serverless computing," IEEE Software, vol. 38, no. 1, pp. 68–74, 2021.

[6] "Intel software guard extensions," 2022.

[7] S. Shillaker and P. Pietzuch, "Faasm: Lightweight isolation for efficient stateful serverless computing," in 2020 USENIX Annual Technical Conference (USENIX ATC 20), pp. 419–433, USENIX Association, July 2020.

[8] P. Stuedi, A. Trivedi, J. Pfefferle, R. Stoica, B. Metzler, N. Ioannou, and I. Koltsidas, "Crail: A high-performance i/o architecture for distributed data processing.," IEEE Data Eng. Bull., vol. 40, no. 1, pp. 38–49, 2017.

[9] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis, "Pocket: Elastic ephemeral storage for serverless analytics," in 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), (Carlsbad, CA), pp. 427–444, USENIX Association, 2018.

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

[11] "Prometheus," 2023.

[12] "Thanos." https://thanos.io.

[13] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, Scalable Schedulers for Large Compute Clusters," in ACM European Conference on Computer Systems (EuroSys), 2013.

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

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

[17] M. Zhang, F. Wang, Y. Zhu, J. Liu, and Z. Wang, "Towards cloud-edge collaborative online video analytics with fine-grained serverless pipelines," in 12th ACM Multimedia Systems Conference (MMSys'21), pp. 80–93, 2021.

[18] P. Stuedi, A. Trivedi, J. Pfefferle, A. Klimovic, A. Schuepbach, and B. Metzler, "Unification of temporary storage in the NodeKernel architecture.," in 2019 USENIX Annual Technical Conference (USENIX ATC 19), 2019.

[19] "Hadoop filesystem implementation for GEDS." `https://github.com/IBM/GEDS`.

[20] W. Jakob, J. Rhinelander, and D. Moldovan, "pybind11 – seamless operability between C++11 and Python." `https://github.com/pybind/pybind11`, 2017.

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

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

[23] 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.

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

[25] 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.

[26] Bytecode Alliance, "The WebAssembly Micro Runtime." `https://github.com/bytecodealliance/wasm-micro-runtime`, 2020.

[27] D. Rivas, F. Guim, J. Polo, P. M. Silva, J. L. Berral, and D. Carrera, "Towards automatic model specialization for edge video analytics," Future Generation Computer Systems, vol. 134, pp. 399–413, 2022.

[28] A. Palmer, P. Phapale, I. Chernyavsky, R. Lavigne, D. Fay, A. Tarasov, V. Kovalev, J. Fuchser, S. Nikolenko, C. Pineau, M. Becker, and T. Alexandrov, "FDR-controlled metabolite annotation for high-resolution imaging mass spectrometry," Nature Methods, vol. 14, pp. 57–60, Jan. 2017.

[29] M. Sánchez-Artigas and G. T. Eizaguirre, "A seer knows best: Optimized object storage shuffling for serverless analytics," in Proceedings of the 23rd ACM/IFIP International Middleware Conference, Middleware '22, (New York, NY, USA), p. 148–160, Association for Computing Machinery, 2022.

[30] IBM, "IBM Topics: What is geospatial data?." `https://www.ibm.com/topics/geospatial-data`, 2022.