

HORIZON EUROPE FRAMEWORK PROGRAMME

# CloudSkin

(grant agreement No 101092646)

## Adaptive virtualization for AI-enabled Cloud-edge Continuum

### D2.3 CLOUDSKIN Architecture Specs and Early Prototypes

Due date of deliverable: 30-06-2024  
Actual submission date: 28-06-2024

Start date of project: 01-01-2023

Duration: 36 months

## Summary of the document

<b>Document Type</b>	Report
<b>Dissemination level</b>	Public
<b>State</b>	v1.0
<b>Number of pages</b>	77
<b>WP/Task related to this document</b>	WP2 / T2.1-T2.4; WP5 / T5.4-T5.7
<b>WP/Task responsible</b>	URV
<b>Leader</b>	Marc Sanchez-Artigas (URV)
<b>Technical Manager</b>	Josep LL. Berral (BSC)
<b>Quality Manager</b>	Raúl Gracia (DELL)
<b>Author(s)</b>	Raúl Gracia-Tinedo (DELL), Sean Ahearne (DELL), Omar Jundi (DELL), Ger Hallissey (DELL), Carlos Segarra (IMP), Peter Pietzuch (IMP), Peini Liu (BSC), Ardhi Putra Pratama Hartono (TUD), Marc Sanchez-Artigas (URV), Josep Calero (URV), José Miguel García (ALT), Bernard Metzler (IBM), María A. Serrano (NBC).
<b>Partner(s) Contributing</b>	All partners
<b>Document ID</b>	CloudSkin_D2.3_Public.pdf
<b>Abstract</b>	<p>Specification of the Architecture and APIs.</p> <p>Documentation, early tutorials, and automated tests for the early prototypes of the different software components.</p> <p>First description and evaluation of the results obtained from use cases' validation using different experiments and workloads.</p>
<b>Keywords</b>	<p>Architecture; Use Cases; Benchmarking; Testbed; Continuum; AI; Orchestration; WebAssembly; Ephemeral Storage; Mobile Computing; Streaming; Metabolomics; Dataspace</p>

## History of changes

Version	Date	Author	Summary of changes
0.1	09-04-2024	Marc Sanchez-Artigas	Structure of the document; First draft of Architecture specs.
0.1	27-04-2024	Josep Calero, Marc Sanchez-Artigas	Description of the metabolomics use case.
0.1	27-04-2024	Raúl Gracia-Tinedo, Sean Ahearne, Omar Jundi	Description of the surgery use case PoC and experiments.
0.2	30-05-2024	Peini Liu	Added subsection on Learning Plane prototype and mobility use case.
0.2	30-05-2024	Javier Santaella	Added subsection "Castelloli infrastructure" on mobility use case.
0.2	03-06-2024	Carlos Segarra, Peter Pietzuch	Initial text for WP4 (C-Cells).
0.2	06-06-2024	Bernard Metzler	Added subsection on GEDS prototype
0.3	20-06-2024	Carlos Segarra, Peter Pietzuch	Update S5 with missing diagrams after M18 meeting.
0.4	20-06-2024	Jose Miguel Garcia	Description for Agricultural Use Case.
1.0	25-06-2024	Marc Sanchez-Artigas	Final version.

## Table of Contents

<b>1</b>	<b>Executive summary</b>	<b>4</b>
<b>2</b>	<b>Introduction</b>	<b>5</b>
2.1	Main innovations . . . . .	5
2.2	Purpose of this document . . . . .	5
2.3	Means of verification . . . . .	6
<b>3</b>	<b>Architecture specifications</b>	<b>7</b>
3.1	Global Architecture . . . . .	7
3.2	Execution Workflow . . . . .	9
3.3	Where is the AI? A Distributed Learning Plane . . . . .	10
3.4	Functional Specifications . . . . .	11
3.5	Software components. . . . .	14
<b>4</b>	<b>Early prototypes</b>	<b>15</b>
4.1	Platform prototypes . . . . .	15
4.1.1	Learning Plane (LP) prototype . . . . .	16
4.1.2	C-Cells prototype . . . . .	17
4.1.3	GEDS prototype . . . . .	18
4.2	Use case prototypes . . . . .	19
4.2.1	Nearby Orchestration Platform . . . . .	19
4.2.2	Lithops Serve . . . . .	26
4.2.3	Pravega Streaming for NCT . . . . .	27
4.2.4	Agricultural Dataspace . . . . .	29
4.2.5	Granny: Granular Management of Scientific Applications with C-Cells . . . . .	30
<b>5</b>	<b>Use cases</b>	<b>32</b>
5.1	Use Case: Mobility . . . . .	33
5.1.1	Overview . . . . .	33
5.1.2	Status of the use case at M18 . . . . .	34
5.1.3	Why this use case needs the compute continuum? . . . . .	34
5.1.4	Where AI helps in this use case? . . . . .	35
5.1.5	Experiments, KPIs and benchmarks . . . . .	35
5.1.6	Early results . . . . .	35
5.2	Use Case: Metabolomics . . . . .	37
5.2.1	Overview . . . . .	37
5.2.2	Status of the use case at M18 . . . . .	38
5.2.3	Why this use case needs the compute continuum? . . . . .	38
5.2.4	Where AI helps in this use case? . . . . .	39
5.2.5	Experiments, KPIs, and benchmarks . . . . .	39
5.2.6	Objective 1: Early results . . . . .	40
5.2.7	Objective 2: Early results . . . . .	45
5.3	Use Case: Computer-Assisted Surgery (CAS) . . . . .	49
5.3.1	Overview . . . . .	49
5.3.2	Status of the use case at M18 . . . . .	49
5.3.3	Why this use case needs the compute continuum? . . . . .	50
5.3.4	Where AI helps in this use case? . . . . .	50
5.3.5	Experiments, KPIs, and benchmarks . . . . .	51
5.3.6	Early results . . . . .	52
5.4	Use Case: Agriculture . . . . .	58



5.4.1	Overview . . . . .	58
5.4.2	Status of the use case at M18 . . . . .	65
5.4.3	Why this use case needs the compute continuum? . . . . .	66
5.4.4	Where AI helps in this use case? . . . . .	66
5.4.5	Experiments, KPIs and benchmarks . . . . .	66
5.4.6	Early results . . . . .	67
6	<b>Description of testbeds implementation and setup</b>	<b>70</b>
6.1	Testbed for the mobility use case . . . . .	70
6.2	Testbed for the metabolomics use case . . . . .	71
6.3	Testbed for the CAS use case . . . . .	74
6.4	Testbed for the agriculture use case . . . . .	74
7	<b>Conclusions</b>	<b>75</b>

## List of Abbreviations and Acronyms

<b>AEMET</b>	Agencia Estatal de Meteorología (State Meteorological Agency (Spain))
<b>AI</b>	Artificial Intelligence
<b>AoT</b>	Ahead-of-Time
<b>API</b>	Application Programming Interface
<b>AR</b>	Augmented Reality
<b>AWS</b>	Amazon Web Services
<b>C-Cell</b>	Cloud-edge Cell
<b>CAS</b>	Computer-Assisted Surgery
<b>CC</b>	Creative Commons
<b>CFI</b>	Control Flow Integrity
<b>COS</b>	Cloud Object Storage
<b>CPU</b>	Central Processing Unit
<b>CSV</b>	Comma-Separated Values
<b>DHCP</b>	Dynamic Host Configuration
<b>DL</b>	Deep Learning
<b>DOI</b>	Digital Object Identifier
<b>DRAM</b>	Dynamic Random-Access Memory
<b>EC2</b>	Elastic Compute Cloud
<b>ECS</b>	Elastic Container Service
<b>EKS</b>	Elastic Kubernetes Service
<b>FaaS</b>	Function-as-a-Service
<b>FDR</b>	False Discovery Rate
<b>FR</b>	Functional Requirement
<b>GEDS</b>	Generic Ephemeral Data Storage
<b>GPU</b>	Graphics Processing Unit
<b>GUI</b>	Graphical User Interface
<b>HDFS</b>	Hadoop File System
<b>HPC</b>	High-Performance Computing
<b>HPDA</b>	High-Performance Distributed Analytics
<b>JIT</b>	Just-in-Time
<b>JNI</b>	Java Native Interface

<b>JSON</b>	JavaScript Object Notation
<b>K8s</b>	Kubernetes
<b>KER</b>	Key Exploitable Result
<b>KPI</b>	Key Performance Indicator
<b>LCM</b>	Life-Cycle Management
<b>LTS</b>	Long-Term Storage
<b>ML</b>	Machine Learning
<b>MPI</b>	Message Passing Interface
<b>MS</b>	Imaging Mass Spectrometry
<b>NAS</b>	Network-Attached Storage
<b>NAT</b>	Network Address Translation
<b>NDVI</b>	Normalized Difference Vegetation Index
<b>NFS</b>	Network File System
<b>NVMe</b>	Non-Volatile Memory express
<b>OCI</b>	Oracle Cloud Infrastructure
<b>OLTP</b>	Online Transactional Processing
<b>OOM</b>	Out Of Memory
<b>OS</b>	Operating System
<b>PAT</b>	Port Address Translation
<b>PoC</b>	Proof of Concept
<b>PVA</b>	Predictive Video Analytics
<b>QoS</b>	Quality of Service
<b>RAM</b>	Random Access Memory
<b>ROS</b>	Robot Operating System
<b>RTSP</b>	Real-Time Streaming Protocol
<b>S3</b>	Simple Storage Service
<b>SDK</b>	Software Development Kit
<b>SFI</b>	Software Fault Isolation
<b>SGX</b>	Software Guard Extensions
<b>SIAM</b>	Sistema de Información Agraria de Murcia (Murcia Agricultural Information System)
<b>SLA</b>	Service Level Agreements
<b>SLO</b>	Service Level Objectives

<b>SSD</b>	Solid-State Drive
<b>TCB</b>	Trusted Computing Base
<b>TEE</b>	Trusted Execution Environment
<b>TTFB</b>	Time To First Byte
<b>VA</b>	Video Analytics
<b>VM</b>	Virtual Machine
<b>WAL</b>	Write-Ahead Log
<b>Wasm, or WASM</b>	WebAssembly
<b>XLSX, or XLS</b>	Excel Spreadsheet

## 1 Executive summary

This document constitutes the second deliverable of **WP2: Architecture and Software Validation**. It details the architecture of the CloudSkin platform and examines the interactions among its software components to culminate in the creation of a true **cognitive computing continuum**. This document not only describes the core components of the platform, but also presents the early Proof-of-Concept (PoC) prototypes for the four use cases in the project: **automotive, metabolomics, computer-assisted surgery, and agriculture**. The report also outlines the different functional requirements and **KPIs** of the CloudSkin platform, and provides use case-specific KPIs to showcase how the platform simplifies and empowers the use cases with measurable impact. Finally, the deliverable details the Cloud-edge testbeds and provides **early results** of the different technologies.

## 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 realize this idea, this project works across various disciplines in order to develop technologies to integrate and orchestrate the distributed data and compute resources. Novel solutions are needed at various levels, from system design, libraries and frameworks, up to data-driven orchestrators that can react to dynamic data volumes, monitoring tools, multi-site data governance policies, etc.

### 2.1 Main innovations

To cover the above requirements to a large extent, CloudSkin pursues to implement 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 major 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 identity” across the continuum, where legacy software stacks running in data centers and HPC clusters (e.g., MPI) can seamlessly run at remote edges, and the code has not need to be rewritten for the targeted platform at hand. And not less important, this innovation also pursues a high level of security, a critical requirement when processing data off-premises. This will be met 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 instrument the storage infrastructure with hooks that enable optimizing end-to-end performance and other key performance indicators (KPIs). This includes developing novel systems that can cover an even wider range of use cases (e.g., a real-time use case with improved fault-tolerance). 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**.

### 2.2 Purpose of this document

This document describes the architecture of the CloudSkin platform used throughout the project. The system architecture description involves:

- Description of the global architecture
- Means of verification and key performance indicators (KPIs)
- Schematic decomposition of the early prototypes,
- Description of interfaces.

In addition, this document also provides a first description and evaluation of the results obtained from the evaluation of use cases using different experiments and workloads. All these topics are addressed in the following sections in this document.

This document is created with the best knowledge available at the moment of writing. The details however are subject to change, during the course of the Cloudskin project additional knowledge may become available throughout the process of experiment and development. Furthermore, external influences (e.g., to other EU-funded related projects) may require interfaces to change and can lead to new specifications.

### 2.3 Means of verification

The three innovations in the project will be validated in real settings using the following general KPIs:

Table 1: Primary KPIs for validating the CloudSkin platform.

Means of verification & KPIs
<ul style="list-style-type: none"><li>• KPI1: Delivering equivalent performance of instrumented Cloud-edge programs compared with centralized Cloud (<math>\approx 1X</math> performance).</li><li>• KPI2: Reduction of cloud offloading (<math>&gt; 50\%</math>), while amortizing edge resources and saving communication bandwidth.</li><li>• KPI3: Achieving real-time processing in edge data analytics, at least in one use case.</li><li>• KPI4: Cloud-edge cells startup times at least 10% faster than containers.</li><li>• KPI5: Execution of complex software stacks such as MPI, and OpenMP “as is” with Cloud-edge cells at close to native speeds despite virtualization (<math>\approx 1X</math> performance).</li><li>• KPI6: Automatic conversion of legacy applications into confidential TEE-enabled Cloud-edge cells (zero development effort).</li><li>• KPI7: Microsecond data access latency despite virtualization and adaptive scaling.</li><li>• KPI8: Automated data-tiering and allocation with very low impact on performance (<math>&lt;1\%</math>).</li><li>• KPI9: At least 2x acceleration of workload processing with serverless computing.</li></ul>

Each use case can contribute other specific KPIs, but the above KPIs constitute a representative reference validation platform (RVP) for the project.

### 3 Architecture specifications

#### 3.1 Global Architecture

The three main innovations [IN1 – 3] will contribute new software components that will be unified in the CloudSkin smart continuum platform through a **logically layered specification**:

- **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 the 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.

- **L2. Execution layer.** Tapping into the CloudSkin continuum platform, developers will be able to implement general applications capable of spanning the entire 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 execute in any machine in the system. Further, “**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 like Intel SGX [2]. The new virtualization technology built upon WebAssembly [1] is termed **C-Cells**.

- **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, and even execute monolithic applications (e.g., written in MPI). However, supporting the wide variety of Cloud-edge workloads, from monolithic applications, microservices, as well as streaming computations requires a powerful and varied set of storage abstractions. Otherwise, I/O operations such as the ingestion of a stream of compressed video frames, or the maintenance of a shared state, can be definite showstoppers for the performant execution of edge-to-Cloud workloads if not properly handled.

In other words, 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.

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.

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 given in Fig.1. The **orchestration layer** includes the **Learning Plane**, the brain of the CloudSkin platform. It fulfills different AI-related functionalities such as ① capturing state and telemetry information, ② modeling the system, ③ managing the catalog of models, and ④ serving recommendations, predictions and forecasting. Since there is a large variety of orchestrators (NearbyOne by NBC, C-Cells Planner or Lithops), the **Learning Plane** must be able to interface with the different orchestrators in the control plane. Orchestrators are the actuators that can put the recommendations from the **Learning Plane** into action. Examples of typical actions are the provisioning of a new worker, the live migration of a C-Cell between the Cloud and the edge, or the placement of task in the suitable edge server.

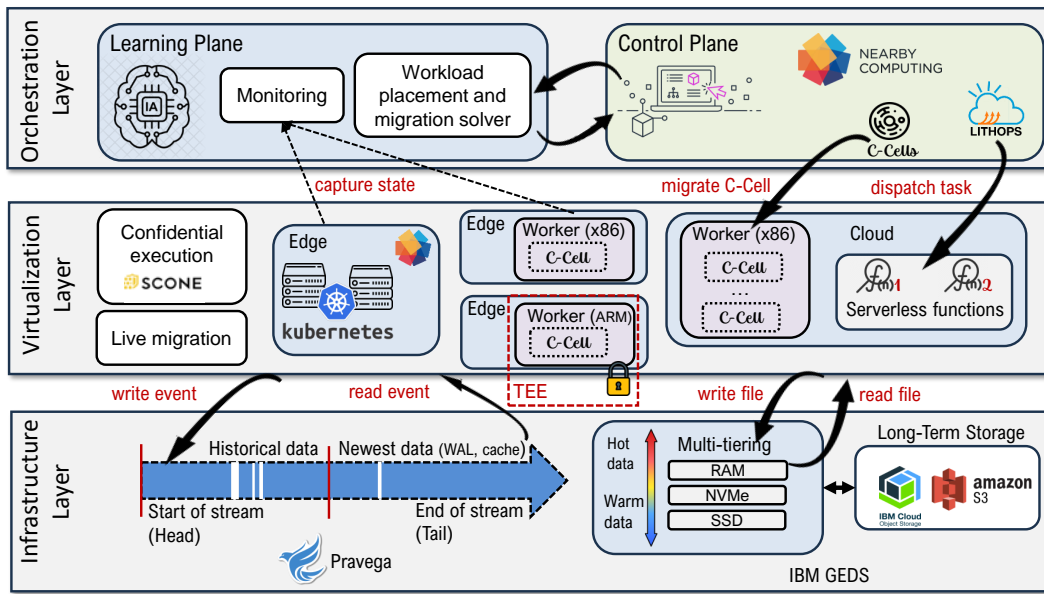


Figure 1: Layered Architecture for the CloudSkin platform.

In the **virtualization layer** lies the worker machines that execute the (WebAssembly-)containerized applications in forms of **C-Cells**, serverless functions, and Kubernetes pods. One important feature of the virtualization layer is ① **Live migration** support: 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. Another key feature is ② **Adaptive virtualization**, which means that optionally and transparently, depending on the data being processed, C-Cells must support hardware-based acceleration (e.g., GPUs) and confidential execution with Trusted Execution Environments (TEE) [3] (e.g., Intel SGX [2]). The CloudSkin platform uses SCONE [4] for this aim.

In the **infrastructure layer** can be found a series of storage services to keep up with data-intensive applications (e.g., Computer-Assisted Surgery). This includes ① **Streaming** workloads, where data streams must be durable, consistent, and elastic, but also ② **Batch** jobs with performance critical I/O operations, such as data shuffling or sharing of data between tasks. As depicted in Fig. 1, CloudSkin handles both types of workloads with the help of Pravega streams and the IBM GEDS storage service. At this layer, we highlight two important features:

- **Elasticity**: is vital to strictly ensure that only no extra storage resources are provisioned to hold the working dataset, a distinguishing feature compared to a classic managed services such as Cloud key-value stores (e.g., AWS S3). Elasticity can be accomplished at multiple levels. This is crucial to real-time applications such as the Computer-Assisted Surgery use case, where the resource pool must be dynamically adjusted to adapt to the current number of surgeries. Since

Pravega streams can change their parallelism on the fly, CloudSkin opens the door to develop proactive autoscaling mechanisms, e.g., using a learning-based forecast model to improve end-to-end latency.

- **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. Therefore, 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 off thin edge devices). Second, it enables the integration of Cloud storage services (e.g., AWS S3) 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.

### 3.2 Execution Workflow

Given the need to accommodate a variety of edge-to-Cloud workloads and applications, the system architecture cannot be tied to a single type of orchestrator. Just in the project, different flavors of edge and Cloud orchestrators coexist together: Function-as-a-Service (FaaS) orchestrators such as Lithops, and edge orchestrators such as NearbyOne [5], specialized to support different types of applications across both on-prem edge and the network edge (network functions, e.g., RAN, distributed core). For this reason, it is important to provide an overview of how an application can typically run under the CloudSkin platform. This requires the following steps:

- ❶ **Package the application into containers or functions:** To execute an application on CloudSkin, developers are expected to package it into one or more containers, WebAssembly binary files, as well as serverless functions. As usual, we assume a container to be a standalone executable package that comprises everything needed to run the application, including the code, runtime, system tools and libraries. In the case of WebAssembly, we recall that WebAssembly is a binary format designed to run on a target architecture and execute the application code in a sandbox, isolated from the host computer, at near-native performance. This means that the system itself must provide the runtime to execute the WebAssembly binaries. This relieves users from such responsibility, but also means that the runtime must be able to run programs of all sorts such as C, C++, Rust microservices, etc. and even monolithic OpenMP and MPI programs. We achieve this with the novel C-Cells runtime. For serverless functions, the language-specific runtime is expected to be provided by the system as well.
- ❷ **Push the code to a CloudSkin supported orchestrator:** This requires to describe the initial state of the application (e.g., the number of replicas of each container) to manage the deployment and scaling of the application. This information is orchestrator-specific. To illustrate this, we notice that Kubernetes uses manifest files. Or Lithops uses simple configuration files in YAML format. Whatever the orchestrator is, it must interface with the Learning Plane through a standardizable API to relinquish AI-enabled decision-making to the Learning Plane.
- ❸ **Expose the application to the Learning Plane:** Once the code is uploaded to orchestrator, the orchestrator relays the application information to the Learning Plane. Equipped with resource usage information, the Learning Plane can forecast QoS values across the compute continuum. Based on these predictions, a heuristic algorithm determines the optimal system for application placement. This can be archived with historical placement records. As more data accumulates, task placement can become more precise to optimize resource utilization, improve performance or curtail monetary costs. Data-driven algorithms such as neural networks are good candidates. However, to get optimal prediction results, the training dataset needs to be pretty good with a low avoidable bias. Also, the training set needs to generalize very well to similar applications to be useful. Whatever the method is, a recommendation is bounced back to the orchestrator.

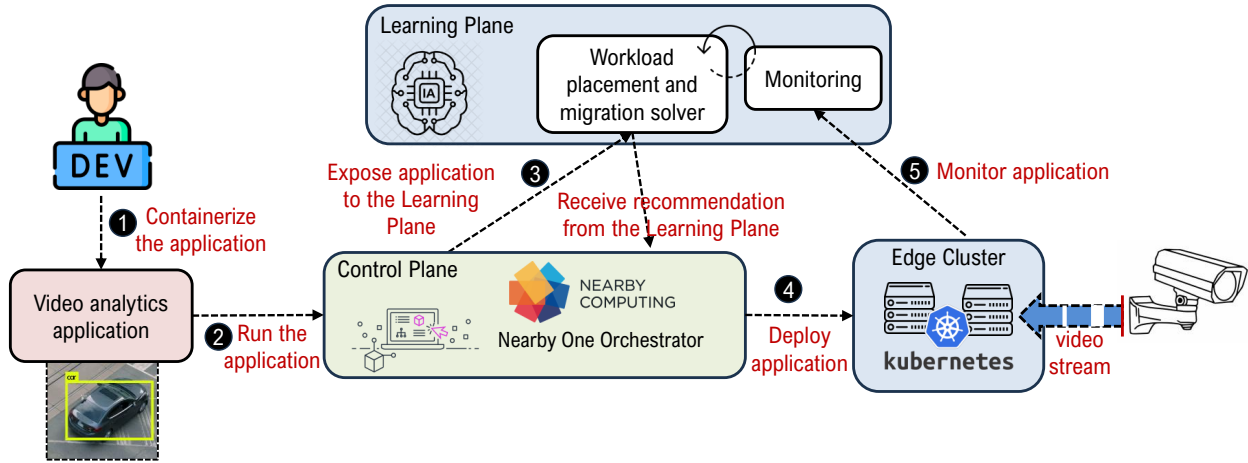


Figure 2: Execution workflow for a video analytics application on CloudSkin.

**④ Deploy the application:** With the recommendation from the Learning Plane, the orchestrator then deploys the application containers, or the WebAssembly binaries, to the optimal platform in the Cloud and the edge. The placement recommendation may be coarse-grained in the sense that the Learning Plane may propose the deployment of an application to a given Kubernetes edge cluster but not indicate to which cluster node. Or it may be fine-grained and exploit node affinity in Kubernetes to constrain placement to GPU nodes or with Intel SGX support.

**⑤ Monitor and manage the deployed application:** After the application has been deployed, the Learning Plane maintains continuous surveillance over both the application and the continuum infrastructure, assessing the need for re-provisioning or live migration. This can occur due to a number of reasons such as the arrival of a new application or due to a change in the application workload. For instance, this can take place in the **Metabolomics** use case due to the arrival of a large number of images to classify, or due to the scheduling of a new surgery in the **Computer-Assisted Surgery** use case.

If a new application arrives, then the steps ① to ④ are followed. Notice that the arrival of a new application may trigger the relocation of a running application to another site to maximize QoS. To handle workload changes, the Learning Plane will regularly try to anticipate application QoS parameters across the heterogeneous continuum systems, leveraging real-time monitoring data to enrich the forecasting process.

These are the general steps to deploy an application on CloudSkin. Depending on the complexity of the application, additional steps may be required, such as configuring storage and networking at the infrastructure layer. However, the above discussion should be a good starting point for deploying applications on the CloudSkin platform. We summarize these steps in the example in Fig. 2 where a video analytics application is deployed using the edge-to-Cloud NearbyOne [5] orchestrator.

### 3.3 Where is the AI? A Distributed Learning Plane

AI models, time-series forecast methods, etc. are planned to be logically centralized in the **Learning Plane**. But this does not mean that the Learning Plane will be centralized in the Cloud. A **distributed Learning Plane** allows a Cloud or a server to combine models from multiple participants, with each participant training their own model locally. For instance, while the decision of whether to relocate an application requires of global knowledge, provisioning local resources for an application can be realized in the Cloud or edge platform where it runs. This allows the natural handling of distributed

data sets, which is a common scenario in many real-world applications, and in some of the use cases in the project. A distributed approach has the following two main benefits:

**Low latency.** For latency-sensitive use cases, enabling intelligence, or at least some parts of it, close to the infrastructure and data sources is essential. This happens in the **Mobility** use case, but also in applications that are characterized by high data velocity, such as the real-time analytics and medical imaging involved in the **Computer-Assisted Surgery** use case. For this reason, parts of the Learning Plane will be migrated to the edge to ensure low-latency processing. This approach will enable novel applications that involve massive data streams that need to be analyzed in a time-critical, secure, and latency-bounded manner.

**Privacy.** Also, deploying intelligence at the edge can enhance privacy by narrowing the extent of data disclosure, especially with the adoption of distributed ML models such as federated learning [6]. This is important for applications that need to address privacy concerns such as the **Computer-Assisted Surgery** use case, which manages health-related information that have to store and manage locally.

However, it also has its **risks**. A distributed Learning Plane also open doors to abuses and attacks. For instance, if a AI model is trained locally at an edge server, and is later outsourced to the Cloud, unauthorized inference on its prediction output should be prevented against sensitive information leakage about the private data used in training the model or against the potential linkage to a private individual based on the sensitive data (e.g., disease surgery) in the prediction output.

In the second half of the project, we will ponder the respective advantages of centralization and decentralization and find suitable tradeoffs to combine the benefits of each type of architecture in the different use cases.

### 3.4 Functional Specifications

Here we provide the main functional requirements (FRs) for the CloudSkin platform. We summarize them in Table 2. FRs associated to each use cases may vary in the second half of the project as early prototypes evolve to MVP (Minimum Viable Product) and include more features. Table 2 reflects the on-going and intended associations between FRs and use case as of M18 of the project.

To best of our knowledge, we see this set of FRs enough to accomplish the mission of CloudSkin, while not falling in the trap of overengineering the solution with unnecessary features that make the system lose generality.

**Further contextualization.** Some FRs in Table 2 need further contextualization within the scope of the project:

- As per FR1, observe that orchestration and management can vary significantly depending on the use case at hand, making specific orchestration stacks a better option than a “one size fits all” approach. For instance, Kubernetes (K8s) [7] can be an ideal orchestrator for Cloud-native apps. Lithops [8] is proficient in serverless multi-cloud orchestration, while other orchestrators such as NearbyOne [5] are specifically tailored for 5G edge scenarios.
- As per FR5, it must be noted that for some use cases, where certain computations do not need to migrate across the Cloud continuum, non-WebAssembly software stacks (e.g., Apache Spark) that are complex to compile to WebAssembly will be leveraged **as is** for better performance.

To span the whole continuum, FR5 also implies 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.

- FR6 is a fundamental functional requirement. Simply put, it qualifies the CloudSkin platform to execute sandboxed code from different tenants within the same container with equivalent semantics to threads and processes, as well as transparently moving it around the continuum in conjunction with FR4. WebAssembly may play here a key role as it has the sufficient generality to support continuum applications, as well as a superior lightness compared to containers [9],

resulting in a powerful tool to run multi-platform software with non-significant performance degradation and small memory footprints.

- Capitalizing on a long experience in building high-performance and ephemeral storage such as Apache Crail [10] and Pocket [11], a new ephemeral storage service called “Generic Ephemeral Data Store”, or GEDS for short, is under development by IBM to provide advanced support for data management across the continuum and thus fulfill FR8.

Table 2: Main functional requirements for the CloudSkin platform at M18.

The four use cases will be labelled as: **Mobility** (stands for “Orchestration of Applications in Cloud-edge and Mobile Scenarios”, **Metabolomics** (stands for “Spatial Metabolomics”), **CAS** (stands for “Computer-Assisted Edge-based Surgery use case”), and **Agriculture** (stands for “Agriculture Dataspace”).

No.	Functional requirement	Software layer(s)	Associated use case(s)	Further context and implications
FR1	Respond and adapt intelligently to changes in application behavior and data variability to optimize where data is being processed (e.g., very close to the user at the edge, or in centralized capacities in the Cloud).	L3	Mobility, Metabolomics, CAS	This will require to interface with orchestrators to offer automatic deployment, mobility and secure adaptability of services from Cloud to edge.
FR2	Ensure extensibility of the AI-enabled control plane with new Machine and Deep Learning models to expand the reach of the CloudSkin platform to other use cases.	L3	All	Interoperability challenges may arise between computing providers, orchestrators and the Learning Plane. Open standards, interoperability models and open platforms should be considered where appropriate.
FR3	Collect and manage metrics and telemetry to extract knowledge from both the underlying infrastructure and the decision-making systems.	All	All	This will require to develop an interface to push telemetry data to the Learning Plane. Standard open-source monitoring and alerting systems (e.g., Prometheus [12]) should be considered where appropriate.
FR4	Enable migration of execution contexts and data in order to facilitate cross-Cloud, Cloud-edge and cross-edge workflow execution to transparently integrate the diverse compute continuum resources.	All	All	Migration of execution contexts and data needs to be lightweight to make relocation transparent to users. Services may be self-migratable, require independence from the provider (FR5) and demand trusted execution (FR7).
FR5	Provide an adaptive virtualization layer that enables the seamless execution of the same legacy code across the whole continuum (e.g, both in an HPC cluster or an at edge server).	L2	Agriculture	This requires the use of portable super lightweight containers that can run anywhere from the edge to the Cloud, e.g., based on WebAssembly [1].
FR6	Virtualize execution memory such that code from different suppliers can safely execute side-by-side in the same physical machine.	L2	Metabolomics	This requirement calls for safety guarantees such as Software Fault Isolation (SFI) to protect computations from security breaches and other types of failures and enforce strict boundaries between collocated processes.
FR7	Ensure confidential processing of data to make users confident that their sensitive data will stay private and encrypted even while being processed in the Cloud and edge.	L2	All	This not only requires the confidential execution of native code, but of lightweight WebAssembly containers to comply with FR5.
FR8	Develop efficient Cloud and edge storage services for efficiently managing ephemeral data.	L1	CAS	To improve I/O performance, the storage service must also support multi-tiering to achieve the targeted performance at the lowest possible cost, as well as to make data survive temporary failures at the edge.
FR9	Integrate an elastic streaming storage fabric to enable edge use cases with stringent low-latency streaming requirements, such as real-time video analytics.	L1	CAS	While auto-scaling mechanisms for stream processing engines exist, elasticity for data streams in the storage is challenging, but it is crucial to adapt to changing data rates.

### 3.5 Software components.

As already stated in D2.1, Table 3 provides the complete list of the software technologies that will be integrated into the CloudSkin platform. For each software tool, this 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 built upon 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 are being reworked to provide the functionality needed to meet the **functional requirements**. One example of this is the new C-Cell abstraction that is being built by reshaping Faasm [9], a high-performance stateful serverless runtime, as stated in deliverable D4.1.

Further, the project represents a great opportunity for partners to add the functionality needed to support Cloud-edge applications into their portfolio offerings (e.g., NBC will integrate the project advances into its flagship 5G orchestration service: NearbyOne [5]).

Table 3: 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.
NearbyOne	Open-source (ETSI); Proprietary license (Nearby Computing)	L3 – Orchestration	Orchestration and automation tool for the edge-to-cloud ecosystem: Infrastructure, Connectivity and Applications
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 (CNCF)	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.

## 4 Early prototypes

Here we discuss the design and implementation of the first prototypes in the project. We distinguish between **platform prototypes** and **use case prototypes**. Concretely, we consider **platform prototypes** to be core components of the platform in the sense that they allow the CloudSkin platform to expand to other scenarios beyond the use cases of the project. On the other hand, **use case prototypes** refer to prototypes that has been designed to address the specific requirements of use cases and combine a subset of the platform prototypes with custom system development. The main reason is to empower the existing software in use cases to harness the combined power of Cloud and edge resources. This has the advantage of favoring “exploitation” by maintaining long software developments in use cases that has already been identified as key exploitable results (KERs). Just to illustrate, the METASPACE platform<sup>1</sup> for spatial metabolomics leverages Lithops [13] as the computing substrate. With the help of the CloudSkin platform, we are extending Lithops to include other computing platforms both in the Cloud and the edge.

Table 4: List of early platform prototypes.

Name	Software Layer	Functional Requirements	KPIs	Short Description
Learning Plane	L3 – Orchestration	FR1, FR2, FR3	KPI1, KPI2	Learning Plane data-connector agent has capabilities to connect orchestrators by writing customized actuator (FR1), to call different ML inference pipelines generated by different use cases (FR2), and to connect telemetry and save predictions data to the database (FR3).
C-Cells	L2 – Execution	FR5, FR6, FR7	KPI1, KPI4, KPI5, KPI6	In D4.2 we present Granny, a first prototype of a distributed system that uses CloudSkin, C-Cells, and C-Cells live migration to improve the performance and utilization of legacy MPI and OpenMP applications when executed in a shared cluster of VMs in the cloud.
GEDS	L1 – Infrastructure	FR8	KPI7, KPI8	The Generic Ephemeral Data Store (GEDS) excels at the efficient handling of temporary data as being created, exchanged, and consumed by compute tasks of a complex, potentially multi-staged computational workload. Efficiency is achieved by direct integration of application buffer management with the lowest tier (Tier 0) of the multi-tiered GEDS.

Table 5: List of deliverables with full details of platform prototypes.

Deliverable no.	Deliverable name	Software layer	Work Package
D5.2	Learning methods for Infrastructure and Workload management	L5	WP5
D4.2	Cloud-edge cells Release Candidate and Specifications	L2	WP4
D3.3	Active Ephemeral Data Store Release Candidate and Specification	L1	WP3

### 4.1 Platform prototypes

To avoid repetition in this deliverable, some of the early prototypes in this section has been deferred to another deliverables as reported in Table 5. For sake of completeness, we provide an overview of the architecture of the platform prototypes in isolation along with their specs. Table 4 summarizes

<sup>1</sup><https://metaspace2020.eu/>



the different prototypes along with the functional requirements that has to fulfill in the CloudSkin platform.

#### 4.1.1 Learning Plane (LP) prototype

The cycle of the Learning Plane **data-connector** starts at the submission of an application to be placed in a certain environment (e.g. Kubernetes cluster, managed by an orchestrator). An instance of the connector (i.e., the LP agent) is configured, indicating where to run, which application or applications to follow, which sources of telemetry are available, which API should it use to communicate decisions and actions to the orchestrator or scheduler, and which models should it use (or train). Then, as an active agent, the connector triggers continuously after a configured time, to retrieve data from the telemetry monitor, generate a forecasting or recommendation, and communicate it to the orchestrator or scheduler.

The implementation architecture of the **data-connector** is shown in Fig. 3. The data-connector acts as a LP agent in charge of providing QoS predictions and making task placement recommendations for the orchestrator. The implementation of the agent is based upon Scanflow-k8s [14][15], where it provides different deployments of model predictions, tracking of metadata and parameters, model registry, and an agent framework for autonomic management. As a result, developers only need to provide custom sensors and actuators depending on each scenario at hand. The detailed description of the learning plane is in deliverable D5.2.

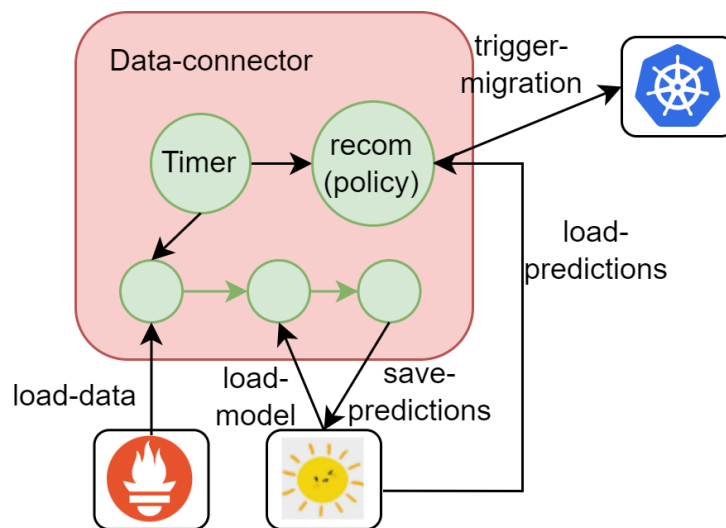


Figure 3: Implementation architecture of the data connector for the Learning Plane.

The Learning Plane works within the orchestration layer as shown in Fig. 1. On the one hand, the data-connector agent can connect custom models (e.g., inference pipeline) to make predictions, and policies for recommendations. On the other hand, the agent actuator can interface with a number of different orchestrators to perform actions. For the first prototype of the data connector, a prototype will be demonstrated for the mobility use case in section 5.1. The next steps will be expand the Learning Plane towards the other use cases in the corresponding measures and applications.

#### 4.1.2 C-Cells prototype

Scientific applications using OpenMP and MPI are common in many domains such as ML, weather forecasting, hydrodynamics, genomics, simulation, and many other domains. To accommodate for larger problem sizes without running out of memory, or taking prohibitively long time's to execute, this application scale up and out (using OpenMP and MPI) to hundreds or thousands of cores across many nodes. In spite of their high-resource needs, scientific applications have not seen widespread adoption in the cloud because cloud resource managers cannot change the parallelism or distribution of these applications once they have started executing.

To enable scientific applications for the cloud, we propose using **C-Cells** to execute threads and processes of multi-threaded and multi-process legacy applications written in OpenMP and MPI, so that they can be run accross the cloud-edge continuum. In addition to continuum enablers, C-Cells has a number of benefits just for the cloud:

- **Performance:** C-Cells execute as fast as native threads and processes, so openMP and MPI application performance is the same; and
- **Zero devops:** C-Cells require no application changes, so the switch is transparent to user code; and
- **Live migrations:** C-Cells can be efficiently migrated at runtime without breaking MPI or OpenMP semantics.

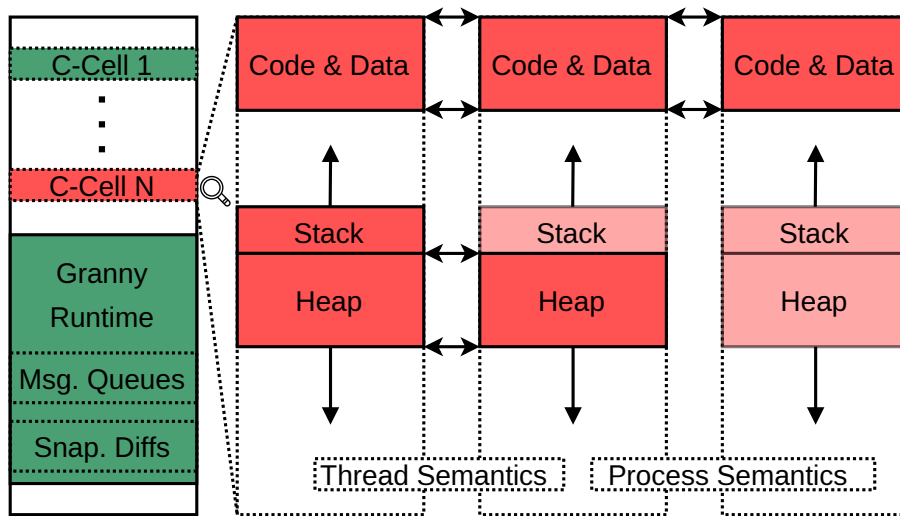


Figure 4: C-Cells on a VM execution within a single virtual address space. Each C-Cell has a simple memory layout that can be used to spawn new child granules with thread or process semantics.

Fig. 4 presents an overview of C-Cells' memory layout when executing threads and processes. C-Cells are co-located in the same address space than the runtime, and they share the static and heap areas depending on whether they are implementing threads or processes. More technical details are given in deliverable D4.2.

### 4.1.3 GEDS prototype

The **Generic Ephemeral Data Store** (GEDS) aims at the efficient handling of temporary data as being created, exchanged, and consumed by tasks of a complex, potentially multi-staged computational workload. Particular attention was paid to the efficient execution of serverless workloads, where the number of compute elements may vary greatly during runtime.

Efficiency is achieved by direct integration of application buffer management with the lowest tier (Tier 0) of the multi-tiered GEDS. Tier 0 interfaces to node local resources through a filesystem interface. This allows direct integration of both local DRAM and local fast storage such as NVMe drives. Since all local GEDS objects are represented as local files, we were able to implement memory mapping for fast object access. Frequent access to local objects is implicitly accelerated by the OS's page cache. With the exception of a namenode maintaining object metadata, the whole data store is implemented as an application loadable library, which avoids running any extra storage service when deploying GEDS. GEDS services are available native, or via Python and Java bindings. At the current implementation status, besides Tier 0, a Persistence Tier provides node-independent, disaggregated storage resources and object persistency, respectively. GEDS configuration allows automated (in case of reaching configurable local resource usage limit) and application-initiated object spilling from Tier 0 into the Persistence Tier. The Persistence Tier has been integrated into GEDS through S3 API. Fig. 5 exemplifies a deployment of GEDS in a Kubernetes environment running a Python workload.

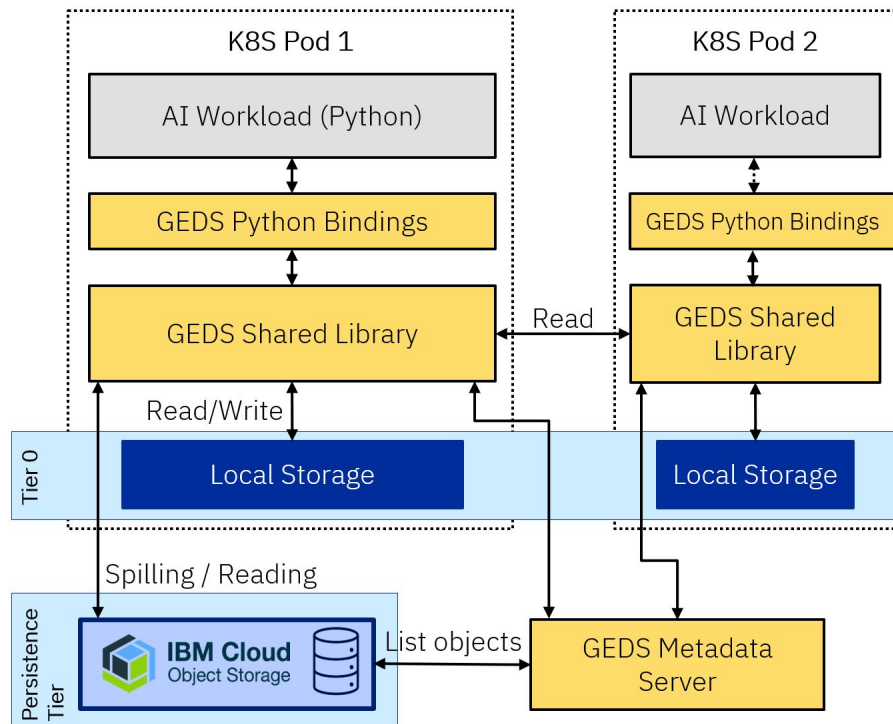


Figure 5: GEDS example deployment in Kubernetes.

The report D3.3 gives a more detailed description of the current functionality and implementation status of GEDS.

At the current stage of the project, GEDS has been successfully integrated with the Computer-Assisted Surgery (CAS) use case, see section 4.2.3, and also report D3.3 for a detailed evaluation. An integration effort of GEDS with the C-Cells workload execution environment has been started.

## 4.2 Use case prototypes

Table 6: List of early use case prototypes.

Name	Use case	Functional Requirements	Short Description
NearbyOne Orchestration Platform	Mobility	FR3, FR4	NearbyOne instance to support the mobility use case.
Lithops Serve	Metabolomics	FR1, FR2, FR3, FR4, FR7	Extends the Lithops framework with a new library to support off-line model serving.
Pravega Streaming for NCT	Surgery	FR1, FR2, FR3, FR9	Provides video streaming service for AI inference workloads for NCT.
Agriculture dataspace	Agriculture	FR5	Provides a platform for data sharing.
Granny: Granular Management of Scientific Applications with C-Cells	Agriculture	FR4, FR5, FR6	Implements scheduling and migration policies for C-Cells executing MPI and OpenMP. We run an MPI application to calculate vegetation indexes.

### 4.2.1 Nearby Orchestration Platform

The NearbyOne orchestrator is responsible for the service onboarding and life-cycle management of cloud-native applications and infrastructure at a global scale, and across the continuum. NearbyOne orchestrator plays the role of the multi-cluster orchestration engine in CloudSkin.

NearbyOne solution provides mechanisms to automate and orchestrate the infrastructure located in Castelloli, and the deployment of components, such as the monitoring stack, the learning plane, or the mobility use case applications. In particular, NearbyOne will manage the set of sites described in Section 4.2.1, i.e., the SE350 edge server and the two virtual machines in the private cloud (control room). Each site has been provisioned with Kubernetes to manage containerized applications. The NearbyOne controller itself has been deployed in a public cloud, in Amazon Web Services (AWS). Overall, the architecture is shown in Fig. 6.

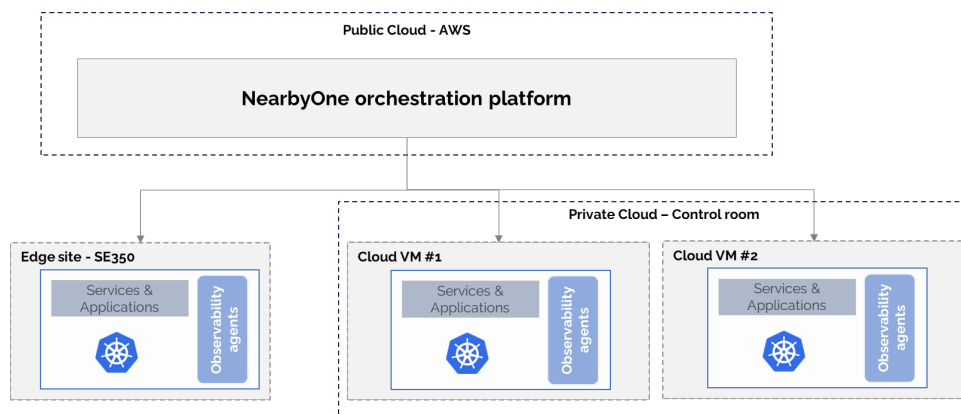


Figure 6: NearbyOne and the orchestrated infrastructure for the mobility use case.

The **mobility use case** application and services are the central component of the use case, and they require the seamless integration of components provided by third-party software developers or vendors. Application developers are not required to make any changes to their application code for

proper integration with the orchestration platform. All configurations are made using declarative languages (YAML). The NearbyOne solution leveraged in CloudSkin is mainly composed of:

- The **NearbyOne Orchestration Platform**, the main component of the solution, is in charge of performing all tasks related to the orchestration of applications and infrastructure.
- The **Nearby Blocks** are distributed components that encapsulate logic and code for different application-specific functionalities.
- The **NearbyOne Observability stack**, built upon cloud-native open-source technologies, and designed to efficiently collect, transport and aggregate telemetry information from the underlying infrastructure.
- The **NearbyOne Northbound Interface (NBI) Orchestration API**, to enable the communication with the Learning Plane .

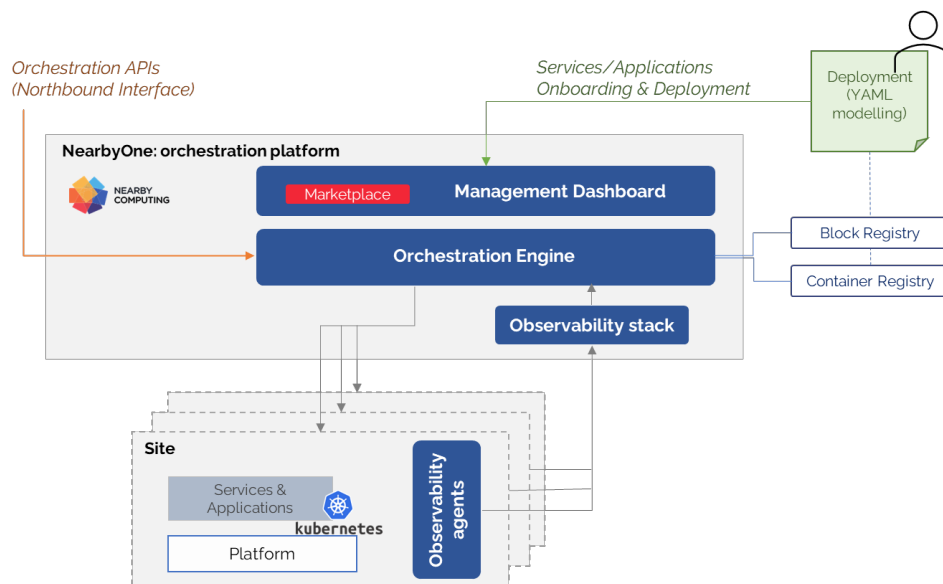


Figure 7: NearbyOne orchestration platform overview.

**NearbyOne orchestration platform.** Fig. 7 shows the components of the NearbyOne orchestration platform. NearbyOne offers a full 360 view of the edge, addressing the major challenges of apps, services, and infrastructure management through a single pane of glass and an intuitive Graphical User Interface (GUI), i.e., the NearbyOne Management Dashboard. NearbyOne not only provides mechanisms to automate and orchestrate provisioning of services but also offers a marketplace that facilitates the choice and chained deployment of services and applications with only one click. The marketplace, easily extensible, gives access to the catalog of solutions available for rapid deployment via the NearbyOne Dashboard (see Fig. 8). Concretely, the marketplace for CloudSkin is composed of:

- The components (Nearby blocks) of the NearbyOne observability stack (described below); and
- The Learning Plane, described in Section 4.1.1 (Nearby block under development); and
- The DL streamer, mobility use case video analytics pipelines; and
- Other example apps, such as NGINX and Torchserve applications.

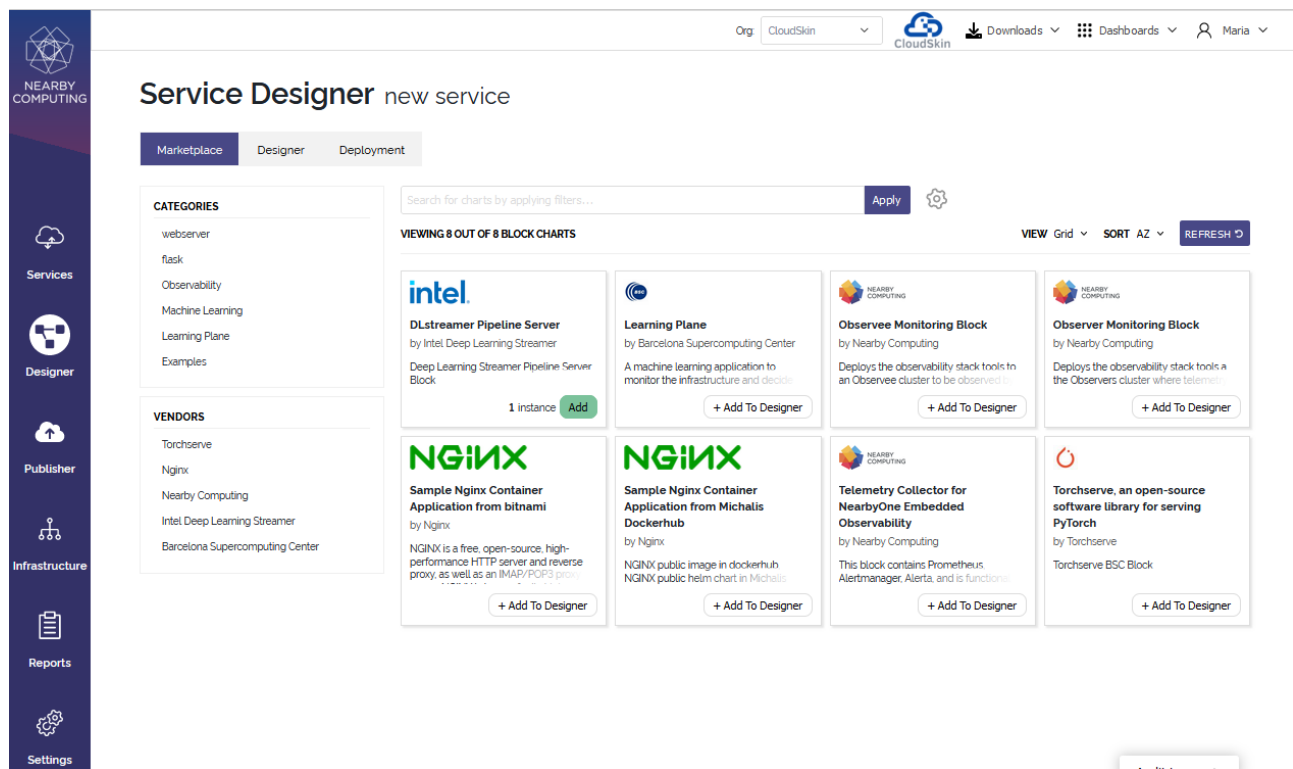


Figure 8: NearbyOne marketplace for CloudSkin.

The **NearbyOne Dashboard** is used to define service deployment chains and establish the relations among them. Deployments, in the form of YAML manifests, contain not only the application deployment information, but also its requirements and placement/configuration policies.

At the core of the orchestrator, the **Orchestration Engine** is the main operational component responsible for managing complex and distributed operations, enforcing eventual consistency rather than transactions. It follows the pattern of self-healing reconciliation loops. It also includes components that manage the policies and other dynamic behavior related to services and infrastructure devices.

Besides the NearbyOne components, other relevant external entities necessary for the operation are the **Block**, **Helm Chart** and **Container** registries, which are the repositories used to store and access application or services artifacts and container images. In the context of CloudSkin, NearbyOne interacts with various types of registries, each serving a unique purpose in the orchestration and deployment of applications.

- **Container Registry:** This is the repository for Docker container images. NearbyOne pulls images from public container registries like Docker Hub<sup>2</sup> or private registries depending on the requirements and security considerations.
- **Helm Chart Registry:** Helm charts are used by CloudSkin's mobility use-case to define, install, and upgrade complex Kubernetes applications. The Helm Chart registry is where these charts are stored. The charts in this registry reference the images stored in the Container registry. CloudSkin mobility use case maintains its own Helm Chart registry implemented under the Harbor open-source project<sup>3</sup>, but it can also interact with public registries that support OCI, such as Docker Hub.
- **Block Registry:** Similar to the Helm Chart registry, CloudSkin mobility use case uses a Block

<sup>2</sup><https://hub.docker.com/>

<sup>3</sup><https://goharbor.io>

registry, too. The Block registry is where the NearbyOne Blocks (see below) are stored, which are higher-level components encapsulating a service or application. Each Block references a Helm chart and, in turn, container images. Mobility use case Block registry is also implemented under Harbor.

By leveraging these registries, NearbyOne ensures a smooth and efficient deployment process, enabling rapid scaling and updating of its services and applications. It is important to note that appropriate access controls and security measures are implemented to protect the integrity and confidentiality of the data in these registries, such as robot accounts for the CloudSkin mobility use case developers.

**Nearby blocks.** In NearbyOne, third-party software components onboarded into the platform to operate services deployment are named Nearby Blocks or simply Blocks. Each Nearby Block contains references to the application logic (service containers in CloudSkin), and they are encapsulated with a set of auxiliary components that provide the means for the application to be effectively managed. Orchestration resources follow a similar syntax to Kubernetes manifests, and Nearby Blocks are packaged and follow a templating syntax similar to Helm charts.

Once the orchestration resources are defined, the orchestrator interact with the underlying Kubernetes infrastructure to deploy the edge services. The orchestrator's Southbound Interface (SBI) plays a crucial role in connecting the orchestrator with the Kubernetes infrastructure. Positioned on the orchestrator's side, the SBI acts as the bridge through which the orchestrator communicates its decisions, intents, and orchestrated actions to the Kubernetes clusters. It provides a standardized and simplified interface, shielding the orchestrator from the complexities of the underlying infrastructure and offering a unified way to convey instructions. This abstraction ensures smooth interoperability, allowing the orchestrator to express its requests without getting tangled in the specific details of the Kubernetes clusters.

Within the SBI framework, the orchestrator's Kubernetes API client module initiates requests, defining desired state changes based on received intents and orchestrated decisions. These requests flow through the SBI to reach the Kubernetes API server module, hosted on the master node of each Kubernetes cluster. The server module processes these requests, making changes in the state of Kubernetes resources according to the orchestrated decisions, such as scaling up or down pods, modifying configurations, or implementing other resource adjustments. Importantly, the Kubernetes resources are deployed using the Helm chart packaging and deployment method, adding a layer of simplicity and efficiency to the orchestration process. In essence, the SBI acts as the orchestrator's communication gateway, enabling dynamic orchestration by translating high-level intents into actionable commands within the Kubernetes infrastructure, promoting a responsive and adaptive orchestration framework.

**The NearbyOne observability stack.** Setting up an observability stack on deployed Kubernetes clusters can enable the collection of performance metrics, tracking of system health in application or device level and even the detection of anomalies in real-time. An observability stack consists of tools for real-time system metrics scraping, data storage (short-term or long-term) and user-friendly visualizations. These monitoring technologies not only offer insights into the inner workings of Kubernetes clusters but also enable proactive problem-solving, efficient resource allocation, and ultimately, enhanced reliability of deployed applications.

The main open-source tools employed for the observability stack deployed on Cloudskin are described below (see Fig. 9).

- **Prometheus**<sup>4</sup> is a monitoring and alerting tool that acts as the main monitoring component in the stack. Its seamless integration with Kubernetes makes it a popular choice for monitoring containerized environments. A large number of Prometheus exporters have been developed for the collection of telemetry time-series data of various sources.

---

<sup>4</sup><https://prometheus.io/>



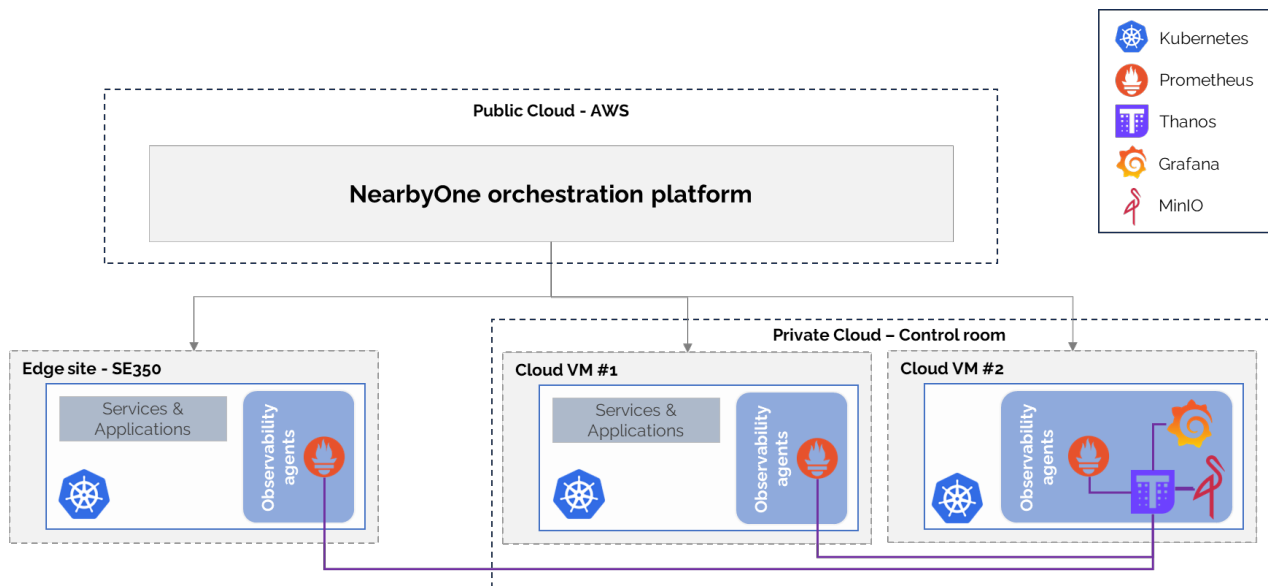


Figure 9: The NearbyOne observability stack deployed for CloudSkin.

- **Grafana**<sup>5</sup> software implements an analytics and visualization platform that turns time-series data into insightful dynamic graphs. It supports a wide range of data backends, including Prometheus, Thanos, Graphite, Elasticsearch, etc., making it highly versatile for monitoring and observability use cases.
- **Thanos**<sup>6</sup> extends the capabilities of Prometheus in terms of scalable, long-term storage and provides a global querying endpoint. With its seamless integration with Prometheus and other monitoring tools, Thanos simplifies the management of large-scale metric data while providing powerful querying capabilities for gaining insights across distributed environments.
- **MinIO**<sup>7</sup> is a high performance, object store for cloud-native and containerized environments, offering scalability, resilience, and simplicity in managing data. In observability use cases it can serve as the telemetry repository of the various scraped metrics.

Taking into consideration the multi-cluster setup of Cloudskin, the proposed architecture divides the clusters into a single **Observer** cluster and multiple **Observees**, as shown in Fig. 9.

The **Observer** cluster (the cloud VM #2 in this case), acts as a control room for the Observability stack, where each of the **Observee** clusters, that is, the SE350 edge servers referred to as the cloud VM #1 and the cloud VM #2, expose their short-term monitoring data scraped by Prometheus. The Thanos API deployed on the **Observer** cluster can respond to queries targeting each of the **Observee** clusters and the **Observer** itself, while also storing long-term data via MinIO. Consequently, the data is only stored in the **Observer** cluster, making it a viable solution when monitoring resource constrained edge clusters.

Both deployment and configuration of the Observability stack are seamlessly available through the NearbyOne orchestrator using NearbyOne Blocks. Concretely,

- An **Observer monitoring block** consisting of integrated Prometheus, Thanos, MinIO and Grafana; and
- An **Observee monitoring block** with Prometheus and a Thanos sidecar connection,

<sup>5</sup><https://thanos.io/>

<sup>6</sup><https://grafana.com/>

<sup>7</sup><https://min.io/>



are available in the user-friendly NearbyOne orchestrator's dashboard, illustrated in Fig. 8. The NearbyOne Blocks can easily be deployed to the desired cluster with configurable options regarding the storage capacity, retention policies and frequency of the collected metrics.

**The NearbyOne Northbound Interface (NBI) Orchestration API.** The NBI orchestration RESTful API facilitates AI-driven orchestration and management of services and applications, providing dynamic service management, automation, and optimization across digital infrastructures. This comprehensive API offers endpoints to automate processes that require human intervention via the NearbyOne dashboard. Concretely, the NBI orchestration API will enable the communication between the learning plane (see Section ) and the orchestrator, to automate processes, such as service migration, based on AI decisions.

The NearbyOne NBI API is under development, following OpenAPI Specification version 3.0.3<sup>8</sup>. It will provide a robust interface for orchestrating service chains across cloud and edge computing environments, facilitating deployment, management, and updates of service chains to ensure dynamic, efficient operations across diverse infrastructure setups. Fig. 10 summarize the NearbyOne API endpoints.

---

<sup>8</sup><https://swagger.io/specification/v3/>

NBI Service Management API

1.0.0OAS 3.0

Provides a robust interface for orchestrating service chains across cloud and edge computing environments, facilitating deployment, management, and updates of service chains to ensure dynamic, efficient operations across diverse infrastructure setups.

[Terms of service](#)  
[API Support Team - Website](#)  
[Send email to API Support Team](#)  
[Apache 2.0](#)  
For additional details, visit our [comprehensive documentation portal](#).

Servers

https://YOURENV.envs.nearbycomputing.com - Access the API using the appropriate environment endpoint by replacing '{environment}'

Authorize

Organizations	Obtain organizations details.	^
GET	/organizations Retrieve details for all organizations accessible to the user	v
Infrastructure	Manage infrastructure interactions including listing of devices and their conceptual organization within sites.	^
GET	/infrastructure/sites/{siteId} Retrieve details for a specific site within an organization	v
GET	/infrastructure/devices/{deviceId} Retrieve details for a specific device within an organization	v
Marketplace	Manage marketplace interactions including listing of marketplace offerings.	^
GET	/marketplace/blockcharts List all available block charts in the marketplace	v
GET	/marketplace/blockcharts/{blockName}/{blockVersion} Retrieve specific block chart details from the marketplace	v
Services	Manage the lifecycle of services from deployment to decommissioning.	^
GET	/services/{serviceId} Retrieve deployed service chain by ID	v
PUT	/services/{serviceId} Update a deployed service chain	v
DELETE	/services/{serviceId} Delete a deployed service chain by ID	v
POST	/services Deploy a new service chain	v
GET	/services Retrieve all deployed service chains	v

Figure 10: NearbyOne service orchestration API (OpenAPI definition screenshot).

#### 4.2.2 Lithops Serve

In this section, we describe Lithops Serve, a brand new library developed in this project that extends the Lithops<sup>9</sup> serverless data analytics framework to support off-line model serving. Lithops Serve has been developed to address the needs of the metabolomics use case. See Section 5.2 for further details.

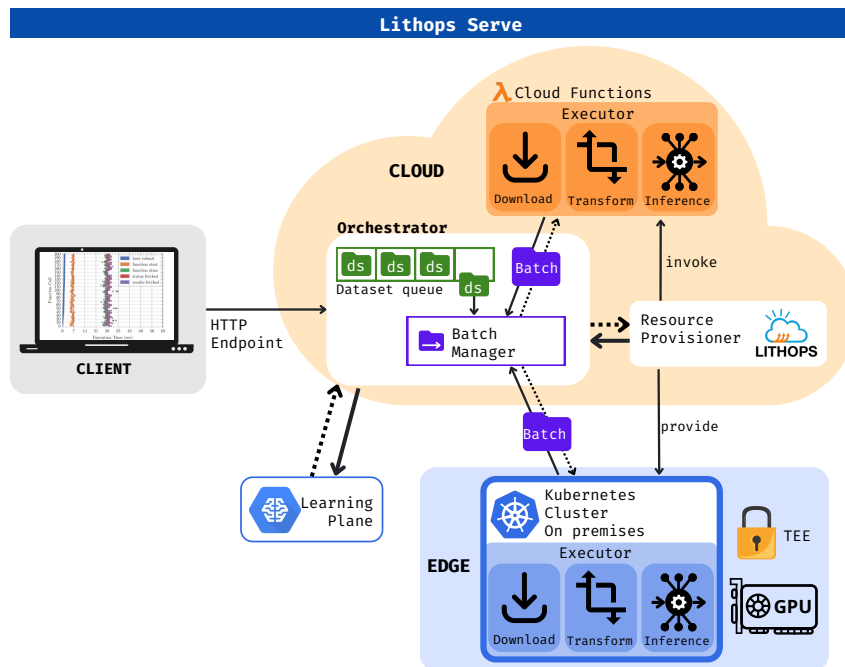


Figure 11: Lithops Serve architecture.

Fig. 11 shows the software architecture of the Lithops Serve PoC. The new model serving system supports the entire DL pipeline for image classification. That is, image loading, transformation, and classification. The PoC follows a declarative programming paradigm, where the code of each task in the pipeline is decorated with a Python decorator of type `@task`. This decorator serves to indicate how each task should execute, for instance, by declaring the degree of parallelism. This is important since tasks may be run on a multi-core architecture and the system should not disallow tasks to leverage multiple CPU threads. A replica of the pipeline is then run within an **Executor** instance. The Executor instance can be of two types: 1.- A serverless function; or 2.- A container. The number of Executor instances of each type is managed by the **Resource Provisioner**, who has the power to scale up and down the number of Executor instances at any given time. The Resource Provisioner is centralized in a cloud server to avoid complex coordination between sites.

A key component of Lithops Serve is the **Batch Manager**, which is responsible for the distribution of the batches images across the running Executor instances. The distribution of images is realized dynamically. This is particularly beneficial because some Executors instances may boot up earlier than others or be faster. With the static scheduling of batches, the faster Executors may finish ahead of time and remain idle for a long time. By dynamically load-balancing the batches, the Batch Manager maximizes resource utilization, saves cost and minimizes end-to-end inference latency. Additionally, it provides fault tolerance by monitoring batch progress and reassigning tasks that fail, or take too long to complete, ensuring robust and efficient operation.

Lithops Serve is ready to integrate with the Learning Plane to optimize resource provisioning, task placement, etc.

<sup>9</sup><https://lithops-cloud.github.io/>

### 4.2.3 Pravega Streaming for NCT

In this early prototype, we have deployed a cluster in Dell's Cork lab with to support a PoC for the NCT use case in CloudSkin (see Fig. 12). The PoC focuses on **containerizing** AI inference models. This is a departure from the existing operation model that data scientists at NCT use nowadays (i.e., script-based models, running AI inference directly on servers). Instead, we provide the necessary tooling for embedding AI inference models in docker containers, making them easy to share and deploy, and connecting them with Pravega [16, 17] through GStreamer [18, 19]. Such a containerized model for running AI video inference on top of Pravega streams does not only mean a productivity boost for managing models and video stream data, but also is more aligned with the cloud-edge continuum model in which applications and data may stretch across heterogeneous infrastructures (see Section 5.3).

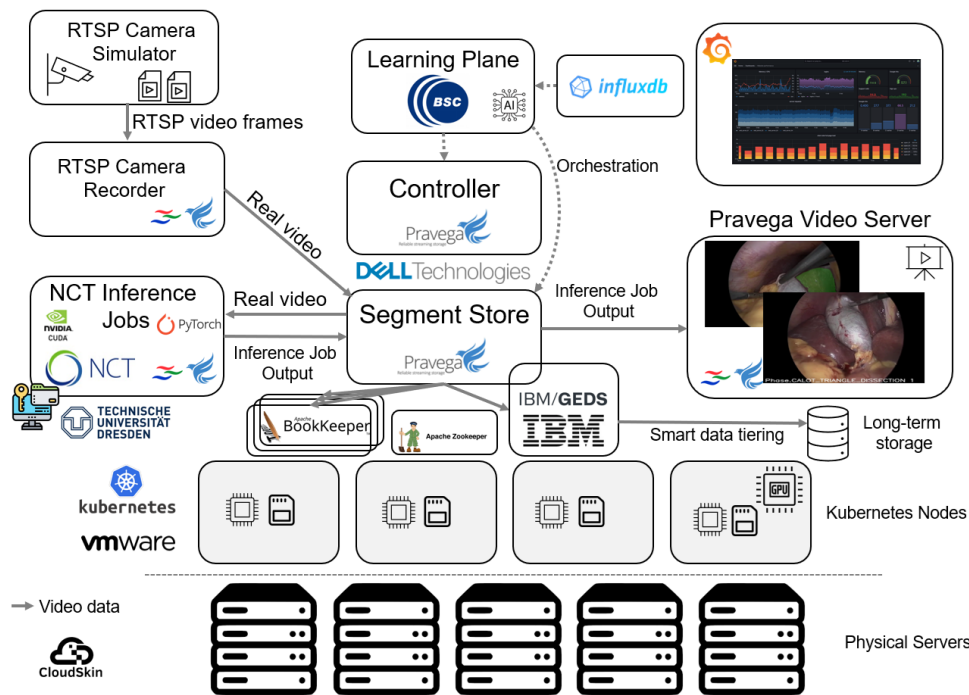


Figure 12: Deployment of Pravega and NCT video stream analytics in Dell's Cork lab.

As shown in Fig. 12, our PoC for the CAS use case consists of several components. Specifically, the main system for ingesting, storing, and serving video data is Pravega. Like GEDS, Pravega is a tiered storage system for data streams that exposes the **stream** abstraction: a unbounded sequence of bytes that achieves durability, consistency, and good performance. In a nutshell, the architecture of Pravega consists of:

- **Controller** instances, which handle metadata and stream lifecycle operations; and
- **Segment Store** instances, which take care of IO; and
- **client libraries**.

A major point of Pravega is that Segment Stores provide **automatic storage tiering**. First, stream data is ingested in a low-latency, durable write-ahead log (WAL), and then the system automatically coalesces stream data for moving it to a high-throughput **long-term storage** (LTS) service. Clearly, this reduces the burden of data management on NCT data scientists, as they do not have to manually move surgery videos from local machines to external storage.

This PoC also integrates Pravega with IBM GEDS for **smart storage tiering** (see deliverable D3.3). In a nutshell, GEDS transparently enhances the storage tiering capabilities of Pravega. Concretely,

GEDS helps Pravega to keep ingesting video streams even under network outages with the external storage service storing stream data (e.g., object storage). This makes the streaming infrastructure more resilient to potential network unreliability across the cloud-edge continuum, without impacting real-time video analytics during surgeries. Our PoC also provides means for storing and visualizing multiple types of metrics about Pravega. In addition to help cluster administration, such metrics can be an interesting substrate for building intelligent resource management. Specifically, this PoC is expected to be integrated with the Learning Plane for taking **auto-scaling decisions** regarding the streaming infrastructure [20] (see deliverable D5.2). Moreover, the foundation of this PoC for containerizing AI inference in NCT may pave the way to explore **confidential computing** models in health-related data (see deliverable D4.2).

Finally, our PoC supports other important video-related features for demonstration purposes. For example, we provide containers that can mimic real video cameras in surgery rooms and a service that helps visualizing GStreamer video streams via HTTP (Pravega video server).

#### 4.2.4 Agricultural Dataspace

At CloudSkin, we have built an edge dataspace to facilitate the sharing and processing of agricultural data.

Initially devised for the edge, data processing has made it necessary to adapt the infrastructure to a hybrid scenario, offloading part of the processing to the continuum. This integration will allow instances to be created dynamically for the execution of intensive tasks, creating a hybrid execution scenario with certain tasks running in the cloud.

In each architectural layer, the most appropriate technology or set of technologies has been chosen for the purposes of the experiment:

- Dataspace development. It is an information management platform for the creation as well as exchange of datasets and projects related to the agricultural sector.
  - In its design, Backpack has been selected. It is a management package designed for the Laravel framework, which provides a series of pre-designed tools and functionalities that accelerate the development process, allowing efforts to be put on the analysis of the needs of the sector.
  - Backpack inherits its security features from Laravel, offering threat protection against SQL injections and Cross-Site Scripting (XSS).
  - Backpack's modular structure makes it easy to add and modify features. This structural advantage has been especially taken into account due to the need for flexibility to adapt to required changes, during the integration, evolution and analysis processes of the platform.
- Development of the public interface. HTML has been chosen for the landing page related to the general information of the project, complemented with CSS and JavaScript for its design.
  - The low execution latency of native HTML improves the user experience by reducing loading times and allowing fluid navigation.
  - Additionally, it is compatible with all modern browsers and devices, ensuring that it is accessible to all users regardless of the platform used.
  - Allows you to easily integrate other technologies and frameworks, allowing the customization and implementation of responsive designs.
- Infrastructure and deployment. Docker.
  - Docker containers ensure that applications work consistently in any environment, whether development, test, or production.
  - Allows you to scale services efficiently by creating multiple container instances. This is essential to dynamically handle increases in demand and maintain optimal performance.
  - Each container operates in isolation, improving security by reducing the risk of conflicts between applications and limiting the scope of potential vulnerabilities.
  - Easily integrates with "orchestration" tools like Kubernetes, allowing you to automate deployments, resource management, and failovers.

The defined architecture facilitates the analysis and execution of performance experiments and adaptation to the continuum. In the hybrid scheme, it will be possible to transfer to the continuum, parts of the heavier functionalities related to data management, allowing by this way that instances can be raised dynamically for the execution of intensive tasks. Fig. 13 provides an overview of the dataspace architecture.

The container where the data space is implemented executes the processes that allow users to define the meta-information, load the datasets, and generate access links to the dataset information. On

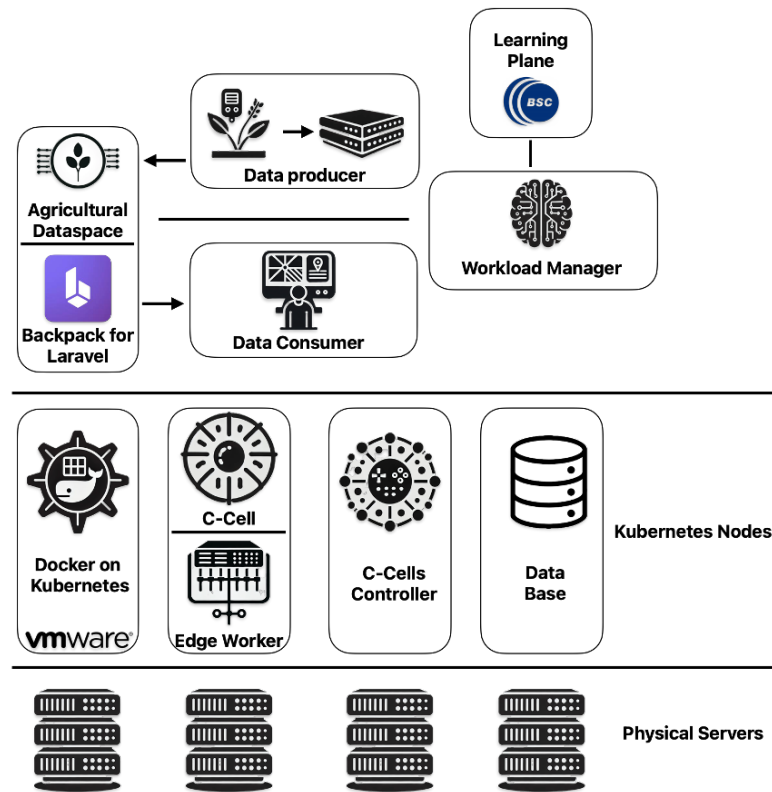


Figure 13: Dataspace architecture.

the other hand, C-Cells technology deployed with Kubernetes as well will serve as a the computation platform for scientific legacy code across the continuum.

Within the project, experiments are contemplated that require the processing of large bursts of small tasks, where the infrastructure improvements of WP3 and the container-like abstraction of WP4 will be of great help. The architecture also includes the C-Cells mechanisms that will provide these improvements. We will also explore the application of AI at the edge to classify crops from the Sentinel-2 time series.

Once we have developed a dataspace on which to manage the datasets, we will focus the second half of the project on finishing the second experiment (water use footprint), where we will implement an analytical application to measure the water use footprint by combining information from sensors (humidity, well water level, etc.) with Sentinel-2 multispectral images and LiDAR data. Part of this pipeline, which is the calculation of vegetation indexes, is currently built with Granny as described below.

New datasets will also be incorporated in this second phase on which tests of use and adaptation of the dataspace to the computational needs of the data will be carried out.

#### 4.2.5 Granny: Granular Management of Scientific Applications with C-Cells

In D4.2 we present Granny, a distributed runtime that uses C-Cells to transparently run unmodified MPI and OpenMP applications. Granny builds on top of the C-Cells prototype and implements a number of scheduling and migration policies to improve the execution of scientific applications in the cloud. Fig. 14 presents a simplified example of a de-fragmentation policy to improve performance of MPI jobs by improving locality as resources in the cluster become available.

In our evaluation in D4.2, we present three different policies: 1. A de-fragmentation policy to change the distribution of MPI jobs; 2. A policy to change the scale of OpenMP applications to use idle CPU cores, and 3. A fault-tolerance policy to run applications on top of cheap, intermittently available, spot VMs.

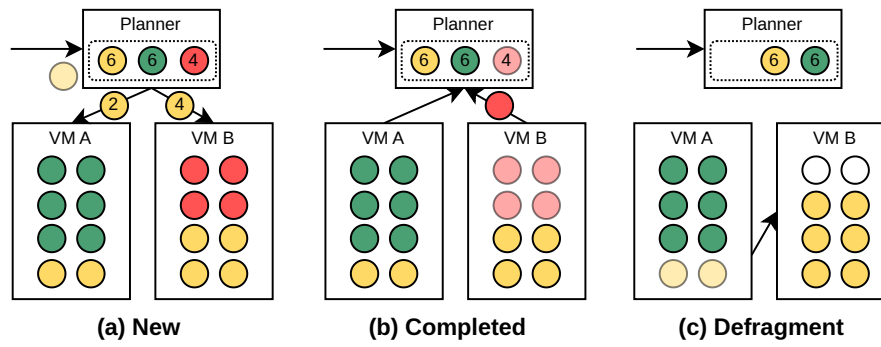


Figure 14: Management using Granny's distributed runtime. Each Granny instance runs on a VM and controls a variable-sized pool of C-Cells. A planner defragments applications to increase locality when other applications have completed.

In this deliverable, we show the applicability of Granny by executing an application from the agriculture domain, a calculation of vegetation indexes (e.g., NDVI) from geospatial images using MPI, and migrating it across the cloud-edge continuum.



## 5 Use cases

Table 7: Summary of the usage of AI in the use cases at M18.

Use case	AI usage	Models
Mobility	<ul style="list-style-type: none"> <li>• <b>QoS-based scheduling and resource provisioning of a video analytics application</b> in cloud-edge environments.</li> <li>• <b>QoS-based migration of a video analytics application</b> in cloud-edge environments.</li> </ul>	AL4DL is used to detect the application patterns; Transform-based model is being explored to generate the workload characteristics; LSTM is trained to predict the nodes and applications behavior.
Metabolomics	<ul style="list-style-type: none"> <li>• <b>Mitigation of cold starts</b>; and</li> <li>• <b>Smart pre-allocation and deallocation of on-premise resources</b> that are slow to boot up and reclaim (e.g., GPU-enabled containers with large serving libraries).</li> </ul>	Still under investigation, models will range from Long Short-Term Memory (LSTM) networks that enables to remember or forget information for long periods, as well as simpler and faster to train models such as Long-Short Term Histograms (LSTHs) [21].
Surgery	<ul style="list-style-type: none"> <li>• <b>Containerized execution</b> of computer-assisted surgery AI models; and</li> <li>• <b>Predictive auto-scaling</b> of streaming infrastructure.</li> </ul>	NCT is using various models for running AI inference in Docker containers instead of using scripts. We are using LSTM and CNN models trained with NCT surgery room utilization traces for auto-scaling Pravega instances.
Agriculture	<ul style="list-style-type: none"> <li>• <b>Automatic resource scaling management</b>, allowing processing of entire datasets; and</li> <li>• <b>Live migration</b> of geospatial tasks.</li> </ul>	The specific models are still under investigation.

Table 8: Summary of use cases KPIs.

Use case	Continuum resources (Cloud, edge, ...)	KPIs (KPIs prefixed with “uc” mean use case-specific KPIs (e.g., ucKPI1))
Mobility	CNX SR650 (cloud); CNX SE350 (edge) both with Kubernetes enabled	KPI1, ucKPI1:QoS; KPI2, ucKPI2:Off-loading
Metabolomics	Amazon Web Services (cloud); Kubernetes on-prem (edge)	KPI9; ucKPI1:Latency; ucKPI2:Throughput; ucKPI3:Performance/\$; ucKPI4:Cost(\$); ucKPI5:AnnotateLatency; ucKPI6:AnnotateCost(\$); ucKPI7:Scalability
Surgery	On-premises cluster (edge) with external storage (core); Amazon Web Services (cloud)	ucKPI1:Productivity; ucKPI2:Latency; ucKPI3:Reliability; ucKPI4:Scalability; ucKPI5:Confidentiality; KPI3; KPI6; KPI7; KPI8;
Agriculture	Docker container at KIO Virtual Data Center	ucKPI1:Apdex Score; ucKPI2:Time to First Byte (TTFB); ucKPI3:C-Cell Execution Overhead; ucKPI4:Cloud-Continuum Execution Overhead

Table 9: Summary of use cases experiments and benchmarks.

Use case	Experiments	Datasets	Benchmarks
Mobility	Experiment 1: Application placement on edge and Cloud.	Application QoS data(i.e., fps, latency), Application and cloud-edge resource usage data(i.e., CPU, memory usage) both 15/05/2024-05/06/2024	Benchmark 1: Application and resource usage data
Metabolomics	Experiment 1: Latency-price relation by dataset size; Experiment 2: Timeline comparison between Lithops Serve functions and ECS containers; Experiment 3: Optimal number of functions in terms of throughput and performance/\$; Experiment 4: Optimal batch size in terms of throughput and performance/\$; Experiment 5: Early sort latency comparison between a VM and serverless functions.; Experiment 6: Early sort cost comparison between a VM and serverless functions.	Objective 1: 2024-02-XX.cloudwatch.log; 2024-02_daemon.log; small.0.5k; small.1k; medium.3k; medium.6k; medium.8k; medium.15k; large.30k; large.35k; large.60k.  Objective 2: X089-Mousebrain (scale factors 1x (small), 3x (medium), 7x (large) and 35x (xlarge))	Benchmark 1: Optimal number of executors in terms of latency and cost; Benchmark 2: Early results of Kubernetes backend with CPUs.; Benchmark 3: Early results of Kubernetes backend with SCONE; Benchmark 4: Throughput and bandwidth capabilities of AWS S3.; Benchmark 5: Early sort performance results of serverless functions on a scaled up dataset.
Surgery	Experiment 1: Deployment of containerized AI models and data management; Experiment 2: Confidential execution of NCT AI models	Cholec80 dataset [22], GStreamer videotestsrc	Benchmark 1: End-to-end video frame latency measurement via GStreamer; Benchmark 2: Induce unavailability of long-term storage while ingesting video data; Benchmark 3: Generate fluctuating video streaming workload and evaluate auto-scaling.
Agriculture	Experiment 1. Validation of the agricultural Dataspace; Experiment 2. Continuum integration analysis.	Agricultural dataset (08/11/2021 – 07/06/2023). Environmental datasets (08/11/2012 - 07/06/2023), (01/01/2023 - 30/09/2023), Infrastructure dataset (04/12/2023 - 06/02/2024)	Benchmark 1: Realistic data from the agricultural environment Benchmark 2: Realistic environmental data Benchmark 3: Realistic infrastructure usage data

## 5.1 Use Case: Mobility

### 5.1.1 Overview

The mobility use case focuses on the distribution of load across cloud-edge environments, on computing nodes with different capabilities and properties, such as high-performance vs. low-power, proximity to data and users vs. close to computing power in the cloud, scalability through nodes vs. distribution across devices, etc. For this reason, the use case proposes a scenario for road-cameras predictive video analytics (PVA), with computing availability next to the road (edge nodes) and next to high-performance (cloud nodes), where edge nodes have lower computational power but are close to cameras and users, while cloud nodes will have higher performance but far from data sources and consumers.

There are two experiments at the mobility use case:

- **Experiment 1. Application placement on edge and cloud:** The first experiment will focus on

deploying an AI-based VA application for vehicle detection using real-time video data from street cameras, the target application can be deployed on the edge and cloud.

- **Experiment 2. Intelligent application migration between edge and cloud:** A second test will consist of migrating dynamically the VA application across edge and cloud resources, according to the client QoS requirements and infrastructure needs (e.g., energy consumption).

Currently, **Experiment 1** has been completed and tested in the CNX infrastructure with realistic data from the circuit camera. Real data has been retrieved from the current testbed in order to train a model for intelligent application migration for **Experiment 2**.

### 5.1.2 Status of the use case at M18

Up to M18, the major efforts of the use case have been devoted on the video-analytics application deployment and the testbed, to enable “**Experiment 1: Application placement on edge and cloud**”. On the one hand, we have been working on the application, in terms of enabling it for video analytics for car detection and testing it in the cloud and edge. For the application, we accomplished:

- Application can properly do video analytics on car camera stream; and
- Application can be placed in the cloud or offloaded to the edge.

On the other hand, we needed to create cloud edge testbeds, with K8s platform running on them. Moreover, we needed to create components within the cloud and edge K8s platform for **monitoring** (NBC’s monitoring), **planning** (BSC’s Learning Plane) and **execution** (NBC’s orchestration). At the time of this deliverable, we have achieved the following:

- Application can be placed by NBC’s orchestration platform on infrastructure; and
- Application and infrastructure metrics can be extracted from the aggregated NBC’s monitoring stack; and
- Learning Plane and orchestration service is properly developed. Currently, the Learning Plane is being tested in BSC’s K8s cluster, and the orchestration service is being tested as an NBC’s Cloudlet.

**Next steps.** As the next steps for this use case, we will work on the integration of the Learning Plane with the NBC’s orchestration service to advise the migration of the application in a dynamic environment. Also, we will explore more QoS metrics such as energy to guide the intelligent workload placement and resource provisioning.

### 5.1.3 Why this use case needs the compute continuum?

The adoption of a cloud-edge continuum is crucial for PVA because it offers a balanced approach to processing the application that combines the benefits of both edge computing and cloud computing.

- **Low-latency, bandwidth saving and privacy benefits from edge computing:** edge computing enables processing the video analytics data closer to the source, which reduces the time taken for analytics and could get real-time results. Additionally, edge computing reduces video data transmission, which reduces bandwidth requirements and data transmission costs and also protects personal or sensitive video data from being transmitted to the cloud.
- **Scalability and flexibility of cloud computing:** cloud computing provides elastic resource allocation and easily scales up and down sufficient computing resources. For instance, when PVA has high user demand, it can request computation resources during peak hours. High-performance computing resources power in the cloud can accelerate the complex processing of the video data.
- **Reliability:** Utilizing a cloud-edge continuum allows for the dynamic deployment and migration of PVA applications across distributed resources based on geographical needs, ensuring efficient response to user requests and events.

#### 5.1.4 Where AI helps in this use case?

Since cloud-edge continuum enables the dynamic deployment and migration of PVA applications across distributed resources, intelligent resource management is critical for application placement. To achieve this, the Learning plane will host and provide machine learning models for decision making on which placement is preferred for video-analytics application, considering the quality of service that can be provided in the different locations according to the capabilities and requirements for the application. The provided underlying technologies for this will be the data connector implementing the predictors and recommenders, using Scanflow as the ML engine, and Kubernetes + NearbyOne for orchestration and scheduling.

#### 5.1.5 Experiments, KPIs and benchmarks

Table 10: Summary of use case-specific KPIs for Mobility.

ucKPI	Description
ucKPI1:QoS	Orchestration of edge apps with matching cloud performance (1x).
ucKPI2:Off-loading	AI video-analytics in the edge with accuracy matching HPC environments (1x) and reduction of cloud off-loading (>50%).

This use case conducts several experiments to show the use case KPIs given in Table 10. All the demo and experiments are running in testbed described in section 6.1. On the one side, we conducted an experiment for PVA application placement on our testbed to demonstrate application offloading and performance KPIs, and on the other side, we used the testbed for benchmarking and collecting data.

The objective of the current demonstration, experiment and benchmark is to show: 1. Application can properly do video analytics on a car camera stream. 2. Application can be placed by the NBC's NearbyOne orchestration platform on infrastructure. 3. Application and infrastructure metrics can be extracted from the aggregated monitoring stack. 4. Application can be offloaded from cloud to the edge (KPI2:ucKPI2) 5. Application offloaded to the edge can keep the performance(i.e., fps) as in the cloud, and maintain the latency constraints (KPI1:ucKPI1).

#### 5.1.6 Early results

**Experiment 1: Application placement on edge and cloud.** This very first experiment pursues the successful deployment of a video analytics (VA) application on the edge and the cloud. In this test, we deployed a VA application through NBC orchestration platform on CNX infrastructure. We enabled fully offloading of the application from the cloud to the edge.

Fig. 15 shows the performance of the VA application measured as frames-per-second (fps) both on the cloud and edge. As shown in this figure, our edge deployment preserves the latency constraints of 100 ms as corroborated in Fig. 16.

*Edge VA apps match the same performance as cloud VA apps; so KPI1/ucKPI1 is fulfilled. That is, the average fps is the same as in the cloud (Fig. 15), subject to a pipeline computing latency of less than 100 ms as verified in Fig. 16.*

*Cloud apps can be fully offloaded to the edge, and the same model reports the same accuracy; so KPI2(40%)/ucKPI2 is fulfilled.*

**Benchmark 1: Application and resource usage data.** The benchmark consists of collecting edge and cloud data towards learning multi-dimensional time-series for the future recommendations of application migration in a dynamic environment. In this benchmark, we explore VA applications and use a stress application to create dynamic environments. For both edge and cloud, we benchmark its resource consumption, as well as the application resource usage and performance (as shown in Fig 17 - Fig 22).

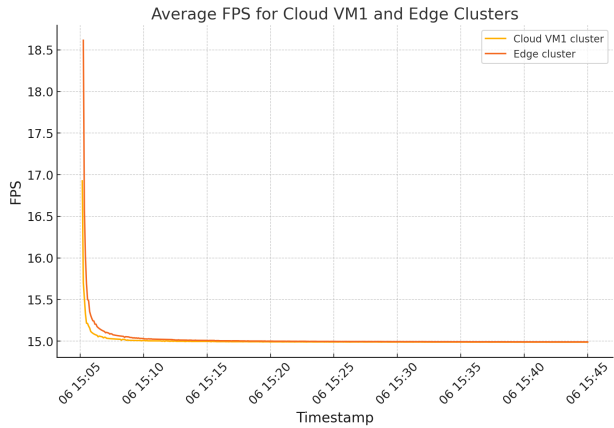


Figure 15: VA application camera video stream avg fps.

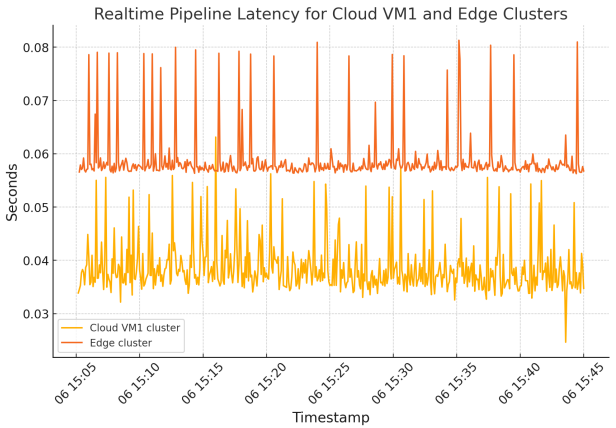


Figure 16: VA application real-time pipeline computing latency.

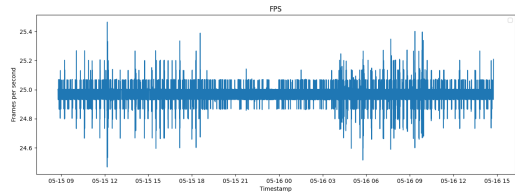


Figure 17: VA application camera video stream avg fps.

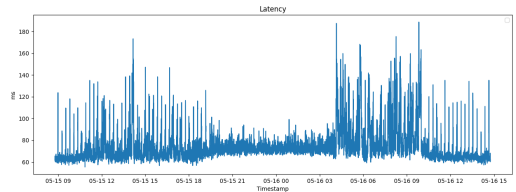


Figure 18: VA application real-time latency.

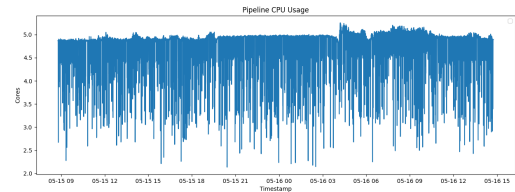


Figure 19: VA application CPU usage.

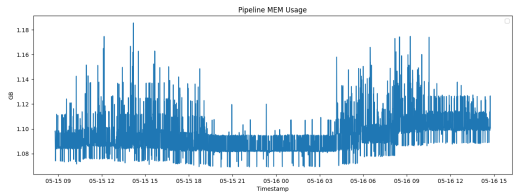


Figure 20: VA application memory usage.

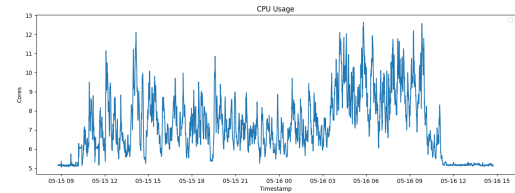


Figure 21: Cloud node CPU usage.

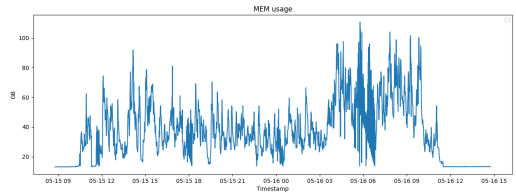


Figure 22: Cloud node memory usage.

## 5.2 Use Case: Metabolomics

### 5.2.1 Overview

A full description of the Metabolomics use case can be found in D2.1. In a nutshell, this use case poses two objectives:

#### 1. Objective 1. AI-enabled optimization of the off-sample ion image service.

METASPACE integrates a service for recognition of off-sample mass spectrometry images with deep learning. Operational on AWS, the existing off-sample service experiences significant idle periods. The workload fluctuates steeply from hour to hour and the existing container-centric implementation of the service suffers from high latency due to long scaling out times. It requires a container to always remain active with the service auto-scaling containers to keep up with the demand.

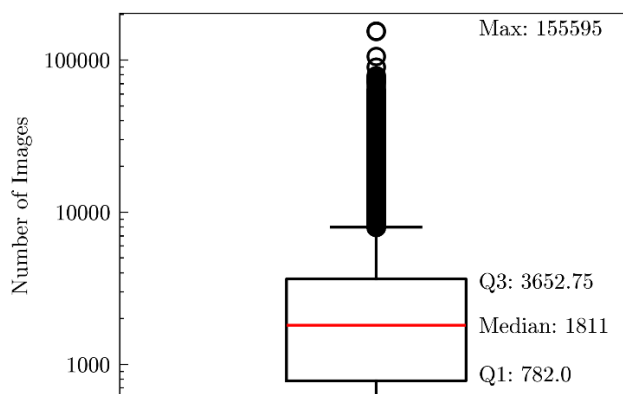


Figure 23: Distribution of dataset sizes in terms of number of images in 2023.

In summary, the goal is to re-implement the off-sample service such that can rapidly scale up to ensure low latency and scale down to zero in the idle periods to save money.

In addition, the new service should be able to run on-premise edge resources with accelerators, as well as to confidentially run inferences using enclaves like Intel SGX to protect confidential datasets of private companies.

#### 2. Objective 2. AI-enabled optimization of stages with data redistribution in the METASPACE molecular annotation pipeline.

In this objective, we aim to optimize the metabolite annotation pipeline itself. Today, METASPACE uses a hybrid implementation combining a large EC2 instance with cloud functions - managed by the Lithops serverless framework. Such hybrid VM-Lithops implementation has been used in production since March 2022 and has enabled a significant increase in scalability. Unfortunately, there is still a bottleneck due to the physical limitations of VMs, namely in the case of a sort stage that runs entirely within the VM instance. Memory prevents METASPACE from processing datasets larger than 250 GB.

The objective is to remove the VM scalability bottleneck, enabling scaling up to hundred-of-GB datasets through an AI-enabled provisioning of serverless functions.

For both objectives, we will base our novel solution upon serverless technology as a cornerstone for resource pooling and rapid elasticity. This will enable the system to respond quickly to workload variations without incurring high latency and large operational costs. We will leverage the Learning Plane to optimize the operation of the service.

### 5.2.2 Status of the use case at M18

Up to month 18, much of our efforts have focused on **Objective 1**. We have already started to tackle the challenges behind **Objective 2** by:

- Examining the code and quantifying the bottleneck.
- Studying how the large EC2 instance can be efficiently replaced by a fleet of serverless functions. Disaggregated storage systems, such as blob storage services, can be used to exchange intermediate data between functions.
- Implementing a preliminary distributed sort operation over serverless functions that would eventually replace the in-place VM-based sort.

Early results of our prior analysis can also be found later in this deliverable.

To accomplish **Objective 1**, we have extended the Lithops<sup>10</sup> framework to support (batch) model serving. The new library is called Lithops Serve. At the time of this writing, Lithops Serve works with AWS Lambda, the Function-as-a-Service (FaaS) platform at AWS and is able to process large datasets in less than 1 minute. The new library upgrades Lithops by installing the code, dependencies and predictive model in a Docker Image.

In the last months, a Kubernetes (K8s) backend was added and battle-tested. The K8s backend is also a critical feature of Lithops Serve in order to enable on-premise edge clusters to the continuum. Very interestingly, the K8s cluster may be equipped with GPU acceleration to perform low-latency inferences.

Further, the executor instances running in a K8s backend may be armored by Intel SGX, powered by SCONE [4]. SCONE is able to lift-and-shift predictive models in a Docker Image effortlessly. The advantage is two-fold: cold start is mitigated and confidential execution is guaranteed by Intel SGX enclave without changing the framework (i.e., PyTorch, Tensorflow). This work is ongoing, since it could not be started until Lithops Serve was ready for K8s deployments in the last two months.

### 5.2.3 Why this use case needs the compute continuum?

The compute continuum is essential for this use case due to the dynamic and unpredictable nature of the workload, where the dataset size in terms of number of images can vary +100 times as shown in Fig. 23.

The METASPACE off-sample service experiences fluctuating workloads throughout the day, with periods of high activity followed by long idle times. This variability in workload demands a compute solution that can scale resources up and down quickly to avoid both over- and under-provisioning. To meet this challenge, we use the Lambda service in the AWS cloud to run inferences at scale. Due to its rapid auto-scaling and cost-efficiency advantages, serverless functions will be the workhorse behind Lithops Serve.

However, commercial serverless platforms such as AWS Lambda lack the support of accelerators and therefore cannot provide low-latency services for large-sized datasets. Similarly, they do not support confidential computing with Trusted Execution Environments (TEEs). For these powerful reasons, this use case will combine serverless functions running in the cloud with on-premise edge resources to achieve:

- **Scalability:** The intelligent split between the cloud and edge will enable the elastic and low-latency processing of workloads by adjusting the pool of resources.
- **Cost-efficiency:** The new solution will be cost-effective by tapping into the pay-per-use model of serverless functions and the available resources on the edge with hardware acceleration (e.g., GPU).

---

<sup>10</sup><https://lithops-cloud.github.io/>

- **Confidential execution:** In the case that a dataset contains sensitive images, e.g., from a private company such as AstraZeneca, the new solution will provide a secure enclave at the edge for image classification while keeping the images encrypted and confidential, even from the host system.

#### 5.2.4 Where AI helps in this use case?

AI plays a crucial role in optimizing the **off-sample mass spectrometry** image recognition service at two levels:

1. **Cold start mitigation.** Cold starts can cause significant performance degradation for serverless functions and containers. Especially for inference serverless functions, the impact is important because it may take much more time to download and load the DL models and serving libraries such as Pytorch than the time to process a medium-sized batch of images.
2. **Optimal resource provisioning.** Another recurrent problem is to find the optimal “sweet spot” between the public cloud and on-prem edge resources. The perfect cocktail can vary depending upon the KPI to optimize: end-to-end latency, cost, or both. For instance, if edge servers were equipped with GPUs, a larger proportion of edge servers might be convenient in order to meet a stringent latency KPI.

By analyzing historical data and patterns, AI can help to train a model that can forecast periods of CPU-intense activity. Armed with this predictive prowess, the platform will be able to anticipate spikes in activity and proactively allocate resources accordingly. By dynamically adjusting the pool of warm functions and on-premise resources by forecast projections, the platform will be able to:

- **Mitigate cold starts;** and
- **Pre-allocate and deallocate on-premise resources** that are “slow” to boot up and reclaim (e.g., Docker containers),

in order to rapidly adapt to the demand. The idea is to maximize the number of serverless executors in **warm state**. When an executor is in a warm state, it is pre-initialized with the classification model and dependencies loaded, meaning it is ready to execute code without further ado.

Concerning **Objective 2**, AI can also play a vital role. Given a dataset of a number of .imz1 files, the right number of serverless executor is another hyperparameter. An overprovisioning of serverless executors can “skyrocket” costs and lead to severe performance degradation due to the saturation of the intermediate storage system. AI can act as an data-driven oracle for this stage of the METASPACE pipeline, pre-provisioning the optimal number of parallel executors that, for instance, optimize the cost-performance ratio.

#### 5.2.5 Experiments, KPIs, and benchmarks

**Objective 1.** A series of experiments and benchmarks were carried to evaluate use case-specific KPIs listed in Table 11 corresponding to the off-sample service (ucKPI1-4). In Section 6.2, we provide the description of the different **testbeds**: the pre-project ECS-based solution in production, and the cloud and on-premises edge testbeds for Lithops Serve.

Several daemons are in charge of data collection. The most important is the update daemon that registers the dataset start and end times. Inside each container, there is a parallel process that stores logs in AWS CloudWatch (the AWS service for monitoring and log collection). All these logs are then extracted after a dataset was classified.

Lithops Serve includes a new trace collection system. Concretely, each executor logs the execution details per batch and per image. Image statistics include the time spent in the load, transformation, and prediction stages. Besides, Lithops has its own log collection system that records function request times, start times, end times, and result fetching times. Together, these logs provide extensive data for plotting and statistical analysis.



Table 11: Summary of use case-specific KPIs for metabolomics.

ucKPI	Description
ucKPI1:Latency	Curtail classification latency by a factor of 10 relative to the AWS ECS-based METASPACE solution.
ucKPI2:Throughput	Achieve a throughput (images/s) that is at least 10 times greater than the existing AWS ECS solution.
ucKPI3:Performance/\$	Achieve at least 2x the performance per dollar compared to the current ECS implementation.
ucKPI4:Cost(\$)	Ensure that the total cost of processing each dataset does not exceed 3x the cost of the ECS implementation.
ucKPI5:AnnotateLatency	Achieve a speedup of at least 1.5X in the sort stage of the annotation pipeline.
ucKPI6:AnnotateCost(\$)	Ensure that the cost of the end-to-end annotation pipeline does not grow more than a 200% despite the use of expensive serverless resources.
ucKPI7:Scalability	Enable the annotation of datasets of 500+ GB.

The system will be subjected to various configurations: number of executors, batch size, etc., in order to reach the different ucKPIs defined in Table 11. The experiments will identify settings that meet the different KPIs. By analyzing the detailed execution data collected by the new trace system, our early results provide valuable insights and rules of thumb that fulfill the targeted ucKPIs.

For benchmarking Lithops Serve, we used 7 datasets. Each dataset was categorized by size (large, medium, small) and the number of images it contains (in thousands), e.g., **small.0.5k** (469 images), **medium.8k** (8,476 images), and **large.30k** (30,068 images).

**Objective 2.** To assess the performance of the new implementation of the sort stage with serverless executors + object storage, we used four datasets labeled by their sizes: **small** (7 GB), **medium** (21 GB), **large** (49 GB) and **xlarge** (250 GB). The benchmarking methodology consists of a comparison between the old solution with a single AWS EC2 instance vs. the serverless multi-executor solution with an equivalent amount of aggregated memory between both implementations. At the time of this writing, our focus is put on two metrics: latency and cost, in order to demonstrate that KPIs ucKPI5 and ucKPI6 are fulfillable.

### 5.2.6 Objective 1: Early results

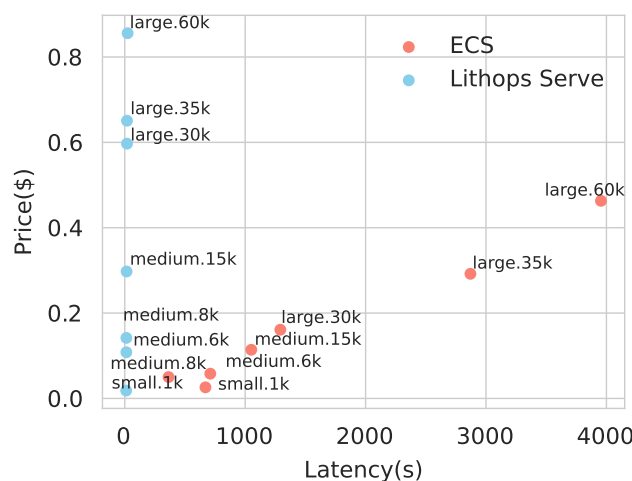


Figure 24: Cost-Performance comparison between Lithops Serve and the METASPACE production solution (ECS).

**Experiment 1: Latency-price relation by dataset size.** The objective is to compare the perfor-

mance of the previous ECS implementation with the new Lithops Serve. To this aim, we used seven datasets of varying sizes using the default setup. In Lithops Serve, the default batch size was set to 32, with each executor instance (AWS Lambda function) processing one single batch.

Results are shown in Fig. 24 as cost-performance scatter plot. As shown in the figure, Lithops Serve demonstrates superior processing speeds across all dataset sizes, showcasing high scalability.

*Processing speeds up relative to ECS are of the order of 20X to 150X, which fulfills by far ucKPI1.*

Also, cost benefits range from being 4X more cost-effective to twice as expensive, depending upon the dataset size and setup at hand. However,

*The cost remains within the budget constraints of ucKPI4.*

**Experiment 2: Timeline comparison between Lithops Serve functions and ECS containers.** The goal of this experiment is to capture the scalability of Lithops Serve against the pre-project ECS-based solution. For this test, we use two medium-sized datasets.

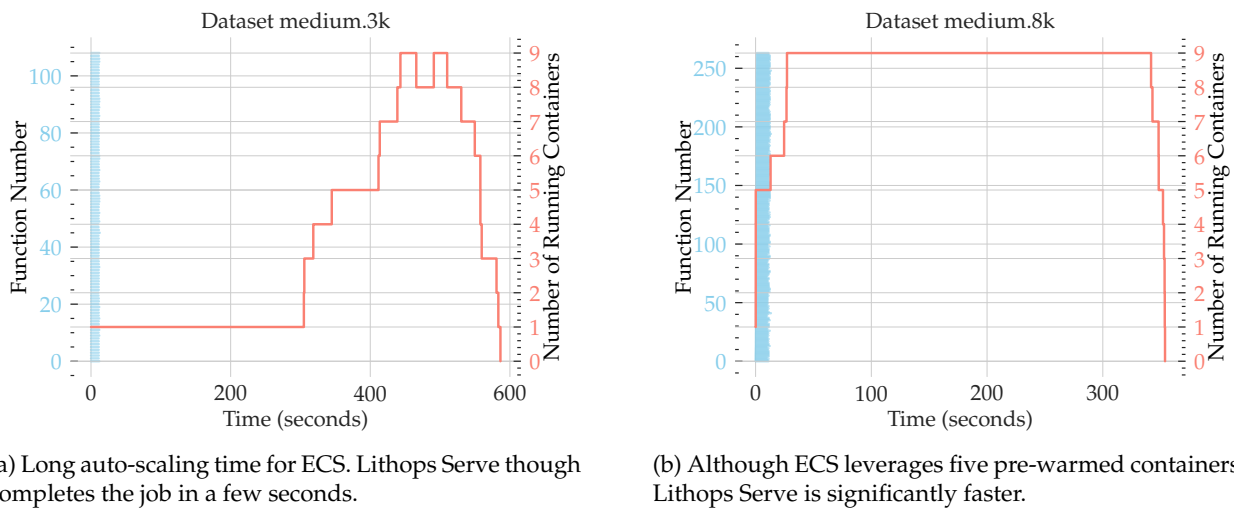


Figure 25: Execution timeline of Lithops Serve Functions (blue) vs. ECS (red).

In Fig. 25, we depict the execution timelines of the Lithops Serve executors on the left side of the subfigures and the ECS containers on the right side. As seen in the figure, Lithops Serve successfully starts up all the executors and completes the job before the first ECS container comes up. In the case of medium.3k (Fig. 25a), a container is consistently active, leading to significant delays in starting new containers. This delay in container creation results in the new implementation taking 60x times less time to complete.

Conversely, medium.8k (Fig. 25b) benefits from pre-warmed containers, perhaps because a dataset was recently under processing. Thanks to the reuse of containers, this larger job terminated before the previous one. In any case,

*The executors in Lithops Serve finished 30x to 60x times earlier, accomplishing ucKPI1.*

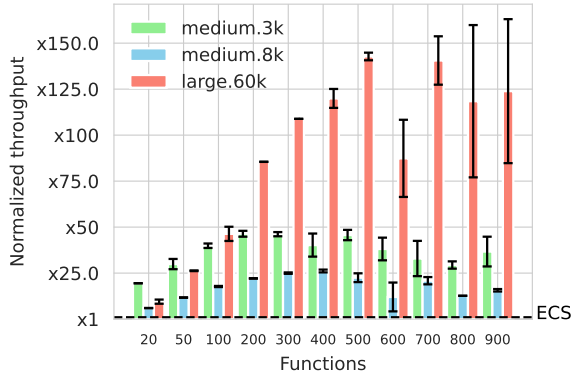
**Experiment 3: Optimal number of functions in terms of throughput and performance/\$.** Fig. 26 reports the execution of three different datasets, each with a batch size of 32 and a variable number of executor in cold state. Each bar in the figure represents the throughput (images/s) or performance/\$ normalized to the corresponding ECS value. Error bars account for standard deviation across various repetitions. From careful inspection, it can be easily seen

*Lithops Serve achieves throughput (images/s) that is 10 to 150 times higher than ECS, successfully meeting ucKPI2.*

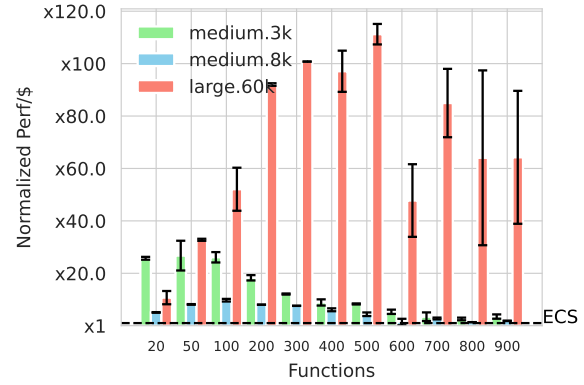
Furthermore, depending on the configuration,

*Performance per dollar is improved by a factor of 5X to 110X, fulfilling ucKPI3.*

Across all datasets, the best performance (images per second) is achieved with  $\sim 500$  functions. However, considering cost-efficiency (\$), fewer functions are preferable for smaller datasets.



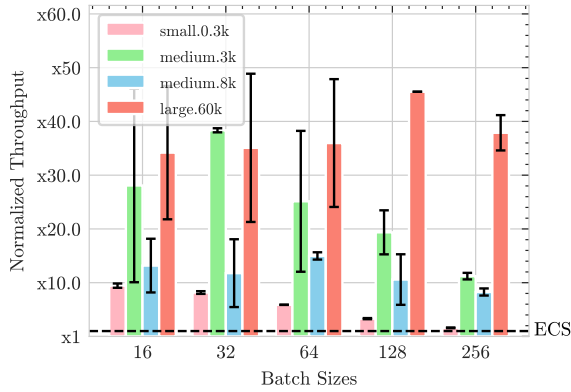
(a) Normalized throughput for an increasing number of functions.



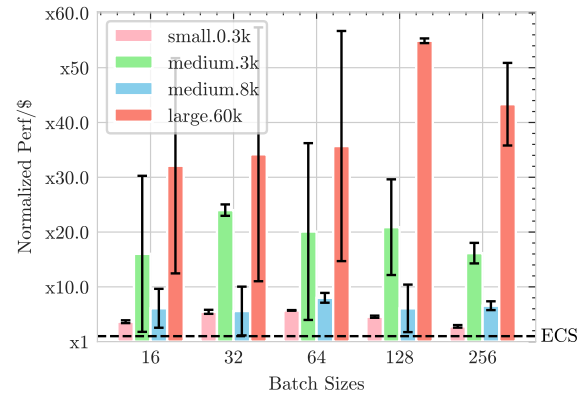
(b) Normalized Performance/\$ for an increasing number of functions.

Figure 26: Impact of the number of executors (AWS Lambda functions) on Lithops Serve performance and cost-efficiency.

**Experiment 4: Optimal batch size in terms of throughput and performance/\$.** Fig. 27 shows impact of the batch size on Lithops Serve. In this experiment, the number of functions (or executors) was limited to a maximum of 100.



(a) Normalized throughput for different batch sizes.



(b) Normalized Perf./\$ for different batch sizes.

Figure 27: Impact of the batch size on Lithops Serve performance and cost-efficiency.

As expected, this figures shows that a smaller batch size is more effective for small datasets. That is, Lithops Serve performs the best with a batch size of 16 for small.03k. For the medium.3k dataset, Lithops Serve maximizes performance for a batch size of 32 images. Likewise, larger datasets benefit from larger batch sizes, with medium.8k achieving peak performance for a batch size of 64 images, and the largest dataset for 128 images, respectively.

Regarding performance per dollar, batch sizes between 32 and 64 are most effective for small and medium datasets, while a batch size of 128 is optimal for the largest dataset. In summary,

*With the right batch size, Lithops Serve fulfills by far ucKPI2 and ucKPI3.*

**Benchmark 1: Optimal number of executors in terms of latency and cost.** Fig. 28 illustrates the utilization of varying numbers of functions for two datasets of different sizes. In both scenarios, a common issue arises: the incremental addition of functions does not consistently decrease latency, but escalates costs. Conversely, provisioning too few functions does not yield cost savings too, but undoubtedly results in slower processing. Such a situation reflects a diminishing returns curve, thus emphasizing the importance of identifying the **optimal number of executors**. For a small dataset of 3,000 images (Fig. 28a), the best configuration typically ranges between 50 and 100 functions, while for a large dataset of 60,000 images (Fig. 28b), it falls within the range of 100-200 executors. To wrap up,

*The key point here is to find the spot where costs begin to rise yet ensuring latency remains low, i.e., Pareto efficiency, to accomplish ucKPI1 and ucKPI2.*

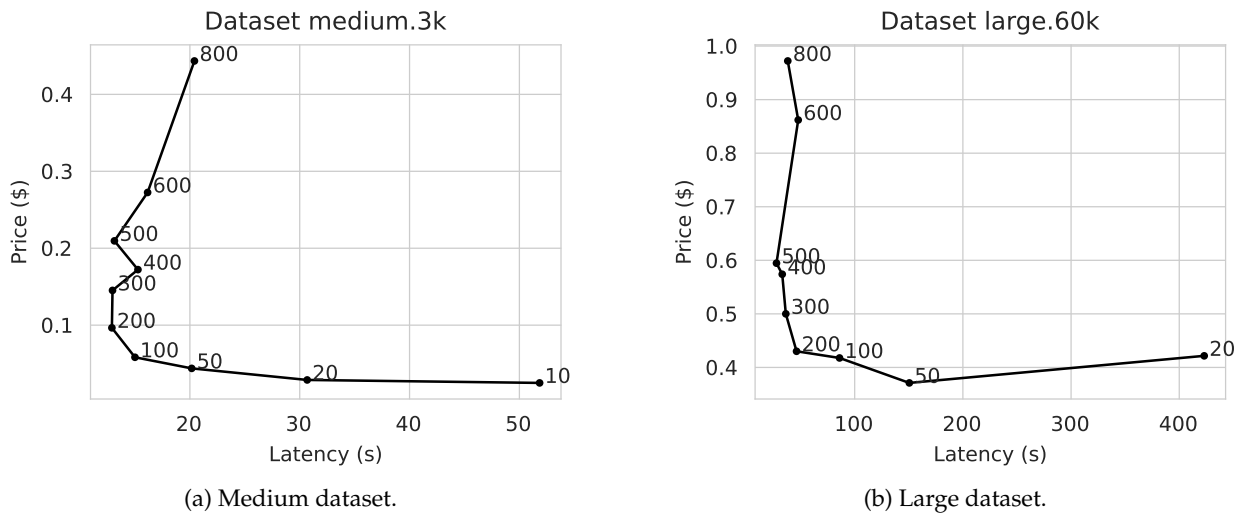


Figure 28: Effect of number of executors on price (\$) and latency (s) for a medium and large datasets. The numbers in the curve refers to the number of executors.

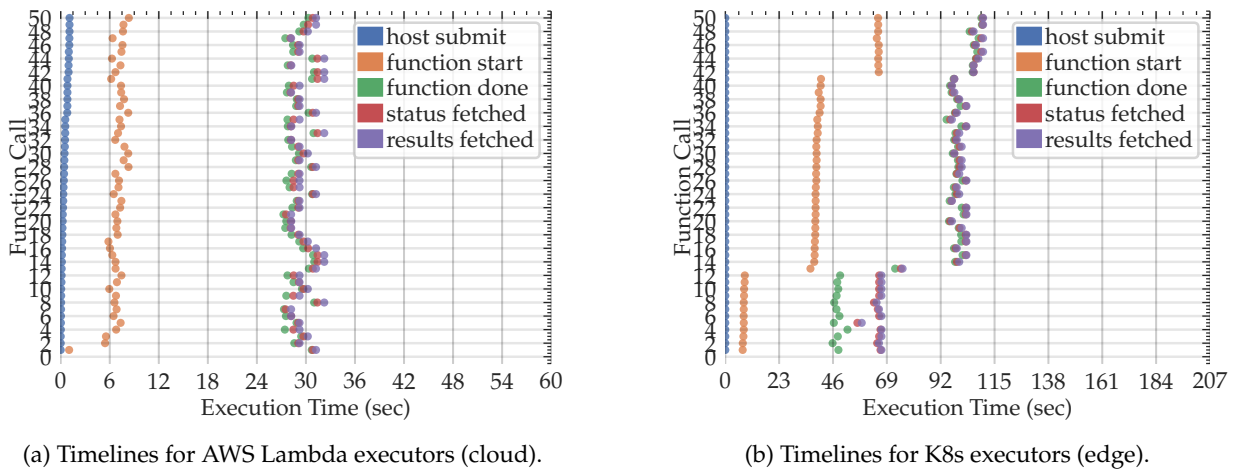


Figure 29: Timeline comparison of AWS Lambda executors (cloud) and Kubernetes executors (edge) on the medium.8k dataset.

**Benchmark 2: Early results of Kubernetes backend with CPUs.** Fig. 29 compares timelines for AWS Lambda executors (cloud) against Kubernetes pods (edge) in cold state. AWS Lambda exhibits minimal cold start times (difference between `host submit` and `function start` times) and a consistent startup pattern. In contrast, Kubernetes pods display more variability in their startup times.

*Kubernetes executors are slow to provision in comparison to AWS Lambda executors. This means that pre-warming containers may be crucial to achieve low-latency inferences. AI-enabled resource provisioning will be key here to mitigate cold starts.*

Fig. 30 displays the throughput for different configurations across four datasets. Our findings reveal that increasing the number of pods does not linearly improve performance. Instead, there is an optimal number of pods for each dataset size beyond which performance degrades. This is likely due to the overhead associated with managing a larger number of pods. For smaller datasets (small.0.3k), fewer pods (30) are needed to achieve peak performance, while larger datasets (medium.15k) require a higher number of pods (50) to optimize throughput.

*As with AWS Lambda executors, there is an optimal number of containers. To meet ucKPI1 on end-to-end latency, AI-enabled resource provisioning will be key here to maximize performance, as well as the split between the cloud and the edge.*

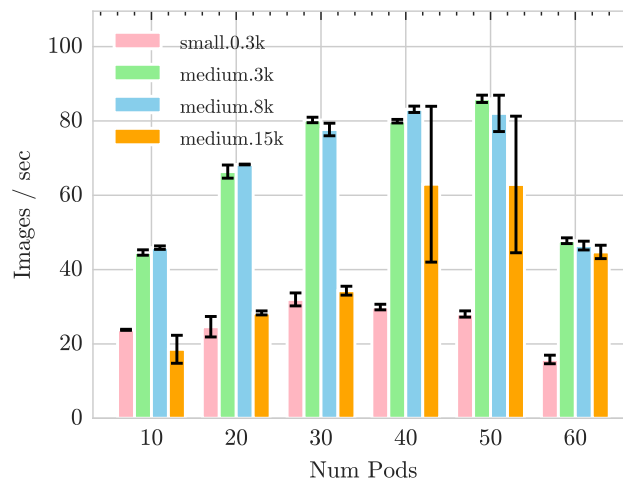


Figure 30: Lithops Serve on the edge. Throughput as a function of the number of K8s pods on four datasets.

**Benchmark 3: Early results of Kubernetes backend with SCONE.** Fig. 31 illustrates the result of running a 16-image batch on two Kubernetes executors across four different SCONE modes: 1. Hardware mode with forking enabled (`scone dbgfork`), 2. Hardware mode without forking (`scone nofork`), 3. Simulation (`sconesim fork`) and 4. native. The results reveal a significant performance degradation in hardware mode, showing an execution time that is roughly 35x slower compared to the other modes. This slowdown occurs regardless of whether forking is enabled or not, indicating that running on the Intel SGX enclave inherently reduces performance. In contrast, the performance of the code in simulation mode is almost identical to that in native mode.

Further tests were conducted using different numbers of executors and batch sizes in three modes: SCONE hardware (`scone fork`), SCONE simulation (`sconesim`), and native. The results are shown in Fig. 32. Unfortunately, the available testbed resources do not support more than 4 executors or batch sizes larger than 16. Even with 4 executors, Out of Memory (OOM) errors were encountered, particularly in hardware mode, as shown in Fig. 32a. Currently, there is no clear pattern indicating the optimal batch size.

*Confidential execution of K8s executors is possible and close to native execution in simulation mode, with some important margin of improvement for the second half of the project.*

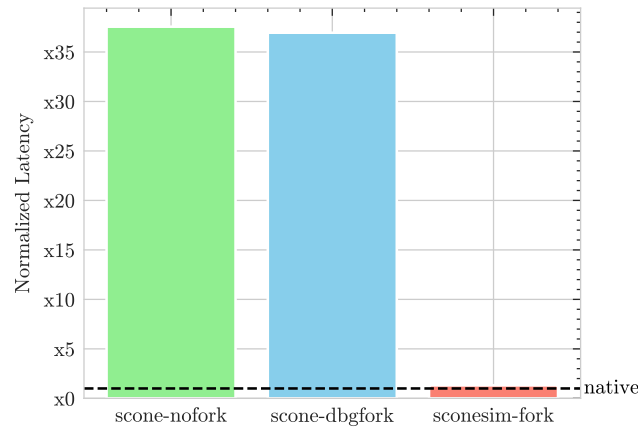
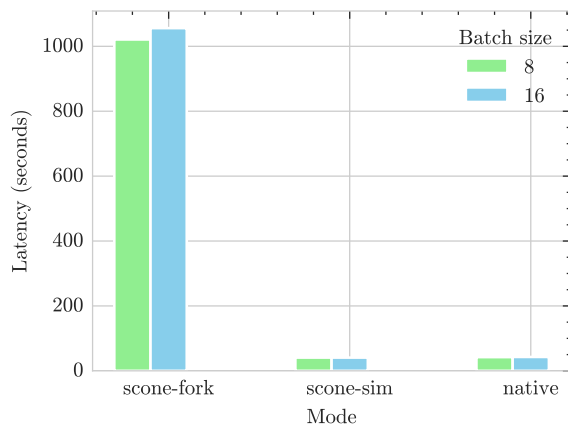
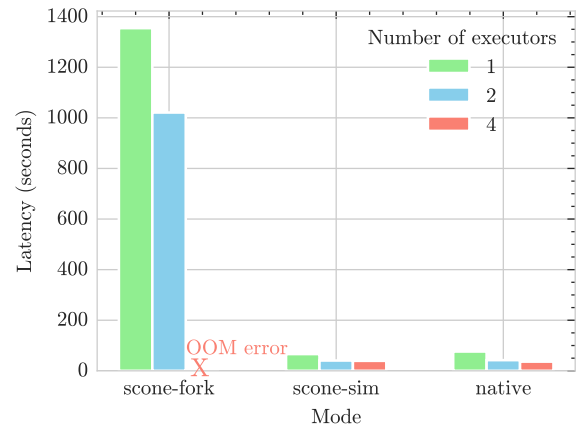


Figure 31: Early test running a batch on 2 executors in different SCONE modes



(a) Timelines for AWS Lambda executors (cloud).



(b) Timelines for K8s executors (edge).

Figure 32: Early test running on different number of executors and batch sizes.

### 5.2.7 Objective 2: Early results

**Benchmark 4: Throughput and bandwidth capabilities of AWS S3.** Sorting at scale is a IO-bound operation that involves a squared number of data partitions to be exchanged between executors. If intermediate storage is the means to facilitate this all-to-all exchange, good performance can only be achieved with high throughput (operations/s) and IO bandwidth (MB/s). To ensure that AWS S3 may be a convenient option to act as ephemeral storage substrate, we first measured its throughput and bandwidth capacities for an increasing number of reader and writer serverless executors.

Fig. 33 reports that throughput scales linearly with the number of executors, both for read and write operations, up to 1,000 serverless functions (the default maximum number of concurrent AWS Lambda instances). Per-client bandwidth increases to around 90MB/s for reads and 85MB/s for writes at 100 concurrent serverless executors. With further concurrency levels, the IO bandwidth per executor remains constant (Table 12).

The lack of bandwidth degradation at higher concurrency levels, along with throughput linearity, makes AWS S3 a promising technology for intensive data exchanges. We can therefore expect AWS S3 to support the levels of parallelism required to sort hundred-GB datasets.

*Throughput and bandwidth of AWS S3 shows scalability on parallel exchange operations, with up to 1,000 concurrent serverless executors, decreasing the odds to become a bottleneck.*

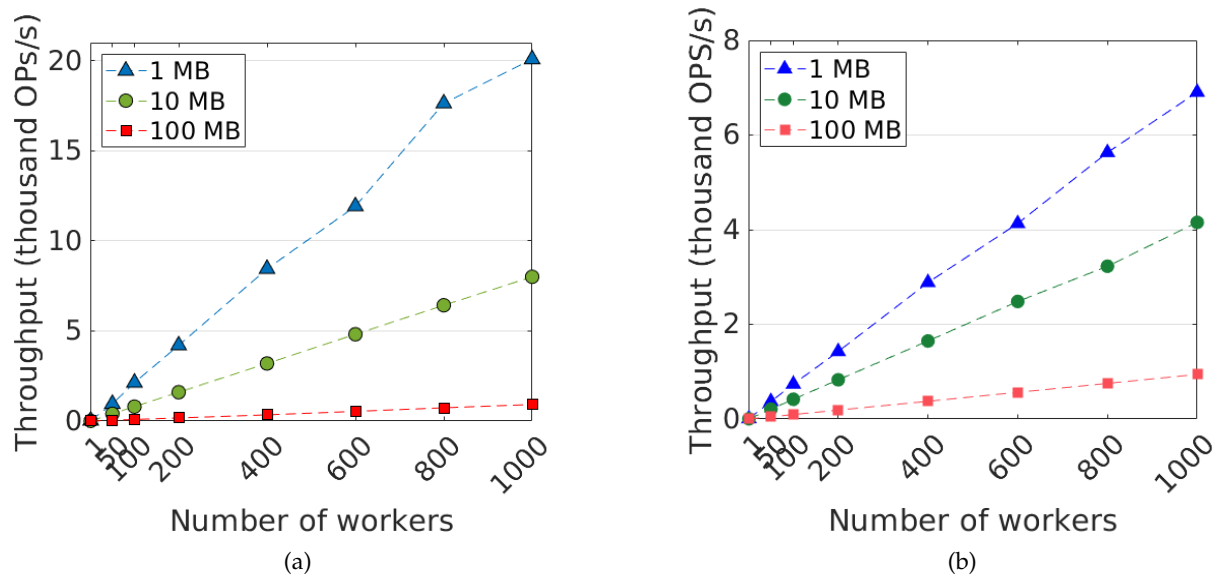


Figure 33: S3 read (a) and write (b) throughput on an increasing number of concurrent clients.

Table 12: Measured per-worker maximum IO bandwidth (in MB/s) for increasing file sizes, in IBM Cloud and AWS S3.

File size (MB)	Read (MB/s)		Write (MB/s)	
	COS	S3	COS	S3
1	35.72	21.80	15.17	7.54
10	62.42	81.89	44.42	42.71
100	62.30	89.72	53.85	86.96

Table 13: Sort configurations, using a VM and serverless functions, for each input.

Input	EC2 Instance	FaaS	
		Mappers	Reducers
small	m4.2xlarge	30	30
medium	m4.4xlarge	90	90
large	m4.10xlarge	210	210

**Experiment 5: Early sort latency comparison between a VM and serverless functions.** In this test, we compared the latency of an in-place sort against a distributed, serverless sort. Table 13 lists the EC2 instance used in each case, which we precisely chose based on the minimal resources needed to process the dataset. We also list the number of parallel serverless functions used in the distributed sort, for which we allocated 1,769MB of memory per function. Sorting is executed over a `imzml` dataset of the sizes described in section 5.2.5, previously converted to bidimensional numpy arrays (including pixel id, m/z value and intensity).



Fig. 34 represents end-to-end sort execution times on each configuration. Latency measurements do not include EC2 startup times. We only measured data ingestion, processing and write time for the VM-based sort. Serverless sort results also include the provisioning time. As shown, the distributed sort is a faster option for the three studied inputs. We reach a speedup of 1.76 at *small*, 2.06 at *medium* and 2.12 *big*, reaching the goals described in ucKPI5.

*Our early prototype is already delivering better latency results than the VM-based solution, thus meeting ucKPI5. These results should be accompanied by a Learning Plane that chooses the right number of functions with the best cost-performance ratio.*

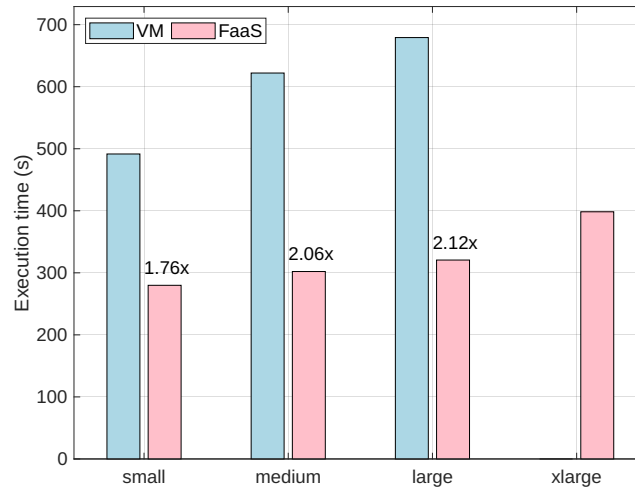


Figure 34: Execution times of a sort operation on different inputs, using a VM vs serverless functions (FaaS). Labels over FaaS bars represent their speedup against their VM counterpart.

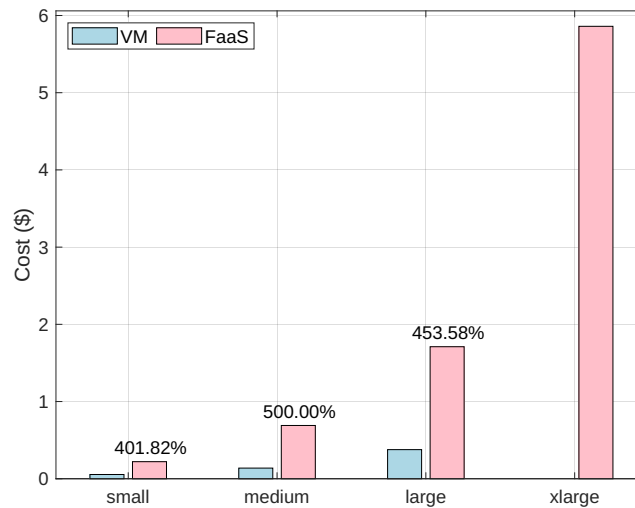


Figure 35: Cost of a sort operation on different inputs, using a VM vs serverless functions (FaaS). Labels over FaaS bars represent their cost increase compared to their VM counterpart.

**Experiment 6: Early sort cost comparison between a VM and serverless functions.** To evaluate the cost of the executions in Experiment 1, we aggregated the AWS S3 requests and the execution time-related costs of each configuration. The early version of the distributed sort does not meet with the requirements of ucKPI6, as depicted in Fig. 35. However, this goal will come through a refined implementation and the addition of an intelligence layer that calculates the cost-minimal configuration under the ucKPI5 and ucKPI6 constraints.



*Enhancing our serverless sort with AI will be key to achieve ucKPI6.*

**Benchmark 5: Early sort performance results of serverless functions on a scaled up dataset.** Finally, we demonstrated that our serverless sort supports currently unfeasible dataset sizes. We sorted a 250GB dataset in 398.40 seconds, including data ingestion and writing back the results to AWS S3. The execution time remains coherent with the magnitudes of the smaller datasets, scaling sublinearly with the input size. We used 2,000 serverless executors (1,000 mappers and 1,000 reducers) with 1,769MB memory each.

*ucKPI7 has not been fully satisfied with our early prototype, but we will evaluate sorting a 500GB dataset in further development stages. The production-level METASPACE annotation pipeline could eventually incorporate a serverless sort to handle large datasets.*

### 5.3 Use Case: Computer-Assisted Surgery (CAS)

#### 5.3.1 Overview

Computer-assisted surgery (CAS) is a rapidly advancing field that leverages technology to enhance the precision, efficiency, and outcomes of surgical procedures. For this, CAS involves integrating various technologies, such as AI, real-time data processing, and advanced imaging, to assist surgeons during operations. By utilizing these technologies, surgeons can gain a deeper understanding of the surgical environment, leading to improved decision-making and reduced risks of complications.

In the context of the CloudSkin, the CAS use case of NCT is designed to support surgeons by providing real-time video analytics, predictive insights, and robust data management systems in the cloud-edge continuum by utilizing the developed frameworks within this project. The primary goal is to create a seamless and efficient surgical process where CloudSkin's advanced technologies aid in various steps of a surgery, from intra-operative guidance to postoperative analysis.

#### 5.3.2 Status of the use case at M18

By month 18, the CAS use case under the CloudSkin project has made significant strides. The efforts have focused on familiarizing with essential technologies, setting up a cluster for the PoC, developing crucial application components, and enhancing the security and portability of these components. .

**Achievements.** The main achievements up to M18 are the following ones:

- **Familiarization with Pravega and GStreamer:**
  - **NCT tech training:** The team has become well-acquainted with Pravega and GStreamer, key technologies for handling real-time video streams and analytics.
- **Video streaming infrastructure for AI inference:**
  - **Setup of CAS PoC cluster:** Pravega and GStreamer have been successfully set up on a cluster for the PoC provided by DELL. This setup is key for running the CAS application and conducting experiments. Besides, DELL has provided the base container image that allows NCT data scientists to integrate AI inference models and running them in a cluster while doing IO via Pravega.
  - **Pravega + GEDS:** Integration of Pravega with IBM GEDS for increasing the tolerance of the streaming surgery infrastructure to unavailability of long-term storage (LTS) while performing stream data tiering. This allows Pravega to keep ingesting data during real-time AI inference in surgeries, even under network disconnections from LTS.
  - **Predictive auto-scaling of Pravega instances:** Adapting to fluctuating workloads requires changing the number of Pravega instances, but adding/removing a Pravega instance may induce a latency spike for a video writer/reader (e.g., disconnection). This may impact the experience of surgeons while doing AI inference on video streams. We propose a predictive auto-scaling model (LSTM-based) for Pravega that minimizes the number of instance auto-scaling events, therefore reducing tail latency in real-time AI inference. This approach paves the way for integrating Pravega with the CloudSkin Learning Plane.
- **Development of AI inference models for CAS tasks:**
  - **Liver Segmentation:** A Python plugin has been developed to assist in segmenting the liver during surgeries.
  - **Surgical Tool Detection:** Another Python plugin has been created to detect surgical tools in real-time.
  - **Surgical Phase Detection for Cholecystectomy:** A plugin for detecting different phases of a cholecystectomy procedure has been developed, aiding in the procedural analysis and guidance.

- **Containerization of the AI models:**

- The entire approach, including the plugins for liver segmentation, surgical tool detection, and surgical phase detection, has been containerized using Docker. This enhances the portability and ease of deployment across different environments.

- **Confidential computing framework:**

- In collaboration with TUD, the Dockerized liver segmentation process has been adapted for secure execution using a confidential computing framework. This adaptation aims to ensure the confidentiality and integrity of the computational processes, although it still needs to be evaluated.

**Next Steps.** The next phase will focus on integrating these plugins and Dockerized components into the broader CloudSkin framework, enhancing real-time video analytics, and leveraging predictive insights for dynamic application migration. Additionally, expanding data sources to include metrics such as energy consumption will be a priority, further optimizing intelligent workload placement and resource provisioning. The evaluation of the confidential computing framework will also be a critical step to ensure secure execution.

By building on these advancements, the CAS use case is poised to leverage the continuum's full potential, significantly improving the precision and efficiency of computer-assisted surgeries.

### 5.3.3 Why this use case needs the compute continuum?

One of the key challenges for the computer-assisted surgery use case of NCT is the ability to execute AI inference on surgery video streams both in real-time and in batch [17]. In terms of infrastructure, these are two completely different data processing approaches with disparate requirements. On the one hand, executing streaming video inference requires **low latency**. To wit, the system is required to minimize the time for a video frame being produced by a camera to serving that frame to an AI inference job, while storing it for durability reasons. The main reason is that NCT is exploiting real-time video inference during surgeries to assist surgeons during the procedure. This is a use case with stringent latency requirements, in which long delays between capturing video and displaying the AI inference output back to the surgeons cannot be tolerated. Therefore, real-time AI inference for computer assisted surgery may be ideally executed at the edge (e.g., servers close to surgery rooms), specially to minimize latency. On the other hand, data scientists at NCT need to perform batch analytics on historical video data from surgeries. In this case, batch jobs like AI model training require **high throughput** for bulk loading historical video data and process it in parallel. This type of activity is more resource intensive and may be better to execute on a larger infrastructure, like a private cloud or large IT infrastructure within NCT's campus. Therefore, we realize that the optimal execution of NCT workloads and tasks encompasses heterogeneous infrastructures, from edge servers close to surgery rooms to larger IT infrastructures for running batch analytics.

In our PoC, containerizing AI inference in NCT brings several advantages. One of them is that now AI inference workloads can be executed on health data across the whole cloud-edge continuum, based on the available resources. However, in this setting, one needs to ensure the confidentiality of both the data and the workload analytics. Trusted Execution Environment (TEE) such as Intel SGX is able to facilitate both challenges. SCONE [4] can lift-and-shift existing applications in a Docker Image effortlessly to be Intel SGX-compatible. The advantage is two-fold: adaptive deployment regardless of the hardware availability and confidential execution is guaranteed by Intel SGX enclave without changing the existing analytical workload.

### 5.3.4 Where AI helps in this use case?

A core goal for NCT is to exploit multimedia (e.g., video, pictures, audio) generated during surgeries in order to apply AI models. Thanks to the different AI models being developed by NCT, surgeons will have a deeper knowledge of the surgery to be performed, both before and while it is taking place. This is expected to dramatically reduce mistakes and complications during surgeries, which is one

of the main reasons for postoperative patient death. To help NCT in this task, CloudSkin proposes a containerized approach for developing and running AI inference models and the exploitation of a streaming storage service for managing surgery data [23]. Before CloudSkin, the daily work of NCT's data scientists involved the execution of AI models via scripts and the manual management of surgery data that is error-prone, time-consuming, and hard to scale. We believe that CloudSkin's infrastructure will improve productivity and help NCT data scientists to be more focused on the AI models they research by simplifying the deployment and data management of video analytics.

Furthermore, AI has other applications in the NCT use case when it comes to the infrastructure. More specifically, we have detected that NCT surgery rooms present strong daily and weekly usage patterns. If we consider that surgery rooms are expected to generate video streams and perform AI inference in real time, this translates into fluctuating workloads with stringent latency requirements. In CloudSkin, we focus on learning and anticipating such workload patterns for proactively adapting the streaming infrastructure via AI/ML techniques [20]. The final outcome will be to integrate such workload prediction techniques for the streaming infrastructure in the CloudSkin Learning Plane.

### 5.3.5 Experiments, KPIs, and benchmarks

In addition to the general KPIs of the project, in the use case of NCT we target the following specific use case KPIs (ucKPI):

Table 14: Summary of use case-specific KPIs for CAS.

ucKPI	Description
ucKPI1:Productivity	The streaming infrastructure should provide means for easily containerizing AI inference models and managing data automatically.
ucKPI2:Latency	To guarantee safety, surgeons should act on up-to-date information, requiring low latency preferably $< 10\text{ms}$ (p95) on HD video streams (25-30 fps).
ucKPI3:Reliability	The real-time AI inference and data ingestion should tolerate network failures with the rest of the NCT campus infrastructure (e.g., minutes).
ucKPI4:Scalability	As the requirements in the operation room changes, e.g. unexpectedly due to a sudden increase of complexity of the surgery or expectantly due to new surgery suites, the algorithm performance should not decrease.
ucKPI5:Confidentiality	When computation is moved to the cloud, patient-specific information should not be accessible.

To validate our contributions on top of the NCT use case, we propose the following experiments and benchmarks:

- *Experiment 1: Deployment of containerized AI models and data management (ucKPI1):* A first goal of our evaluation is to demonstrate that our streaming infrastructure PoC can serve video streams to containerized AI inference models of NCT. Moreover, we want to assess that video streams are automatically being moved to long-term storage, thus abstracting data management tasks from NCT data scientists.
- *Benchmark 1: End-to-end video frame latency measurement via GStreamer (ucKPI2, KPI3):* In the NCT use case, we want to demonstrate that the CloudSkin streaming infrastructure meets the basic performance requirements for real-time AI video inference. To this end, we plan to execute benchmarks with video streams and Pravega [17], the core streaming storage engine of CloudSkin. First, such benchmarks will describe the IO performance of Pravega for synthetic video streams (e.g., write latency, end-to-end latency) for different parameters (e.g., FPS, resolution). Moreover, we will compare the performance of an NCT AI model performing inference on top of a real surgery video stream with the IO performance of Pravega. This would give us a strong evidence of whether our infrastructure for video analytics meets NCT's latency requirements.
- *Benchmark 2: Induce unavailability of long-term storage while ingesting video data (ucKPI3, KPI8):*

While Pravega may be a suitable streaming substrate for video analytics, it was originally designed for cluster environments. This is specially important when we consider the automatic streaming data tiering implemented in Pravega. In a nutshell, Pravega assumes that the long-term storage service to offload stream data will be mostly available. However, in a cloud-edge scenario like NCT, stream data may need to be offloaded from an edge server close to a surgery room to a storage system on another part of a campus (or at another NCT campus). This means that the network connection from the surgery room to the long-term storage service could be unreliable, which would eventually prevent Pravega to continue ingesting data. To overcome this problem, we will experiment integrating an smart ephemeral storage service (IBM GEDS) in Pravega. Concretely, we plan induce outages in the long-term storage system that stores stream data for Pravega and evaluate the improvement in buffering (time, data size) capabilities of Pravega with the integration of GEDS.

- *Benchmark 3: Generate fluctuating video streaming workloads and evaluate auto-scaling (ucKPI4, KPI7):* As we mentioned, we have detected strong daily and weekly patterns in the surgery room utilization at NCT. First, we plan to evaluate whether the streaming infrastructure in CloudSkin (Pravega) can be adapted and auto-scale to accommodate fluctuating workloads [20]. To this end, we will perform a real deployment and induce a trace-based workload based on NCT traces. We will capture metrics like latency after and before the streaming infrastructure auto-scaling events, as well as the latency impact that changing the number of Pravega instances may have on video streams. Moreover, we will evaluate predictive approaches to auto-scaling the streaming infrastructure, measuring aspects like the number of auto-scaling events, the latency distribution of video frames, or the instance time execution, among others. These experiments will elucidate whether Pravega can be an elastic and adaptive streaming substrate for NCT.
- *Experiment 2: Confidential execution of NCT AI models (ucKPI5, KPI6):* Developing a machine learning model is a huge amount of task and integral to the project. One of our goals is to enable the confidential execution of such an AI model effortlessly. We will prepare a Docker image of which is compatible with Intel SGX. Moreover, we plan to integrate such images with the SCONE's framework, offering additional features such as network shield and secret propagations, to name some. Afterwards, we are going to run such model in a trusted execution environment and measure its performance compared to the native one.

### 5.3.6 Early results

**Experiment 1: Deployment of containerized AI models and data management.** The first aspect to address for the NCT use case is the ability to executing containerized AI inference on top of a streaming infrastructure (Pravega) that automatically performs storage tiering of video streams. To this end, we have provided a container base image to NCT that contains the Pravega GStreamer dependencies to connect AI inference jobs. As described in Section 5.3.2, we deployed the "liver segmentation", "instrument detection", and "surgery phase detection" AI jobs as containers. With our PoC in place (see Section 4.2.3), we have been able to ingest video data, read it back via the Docker container, and execute the inference model in a Kubernetes environment. An example of that is Fig. 36 (upper figure), in which we can visualize via the Pravega Video Server the original video stream and a new video stream generated from the output of the liver segmentation AI inference.

Apart from executing containerized AI inference, our PoC also takes care of video data management. To wit, video streams ingested in Pravega are written with low latency to the write-ahead log. Then, Pravega coalesces stream data into chunks and moves them to long-term storage. For NCT, this has two main advantages. First, data scientists do not have to worry about moving video data manually from surgery rooms to an external storage facility, as Pravega does it automatically. Second, Pravega may exploit high throughput in external storage systems, like NFS or object storage, to serve batch analytics jobs (e.g., AI training). As an early result, Fig. 36 (lower figure) shows an experiment

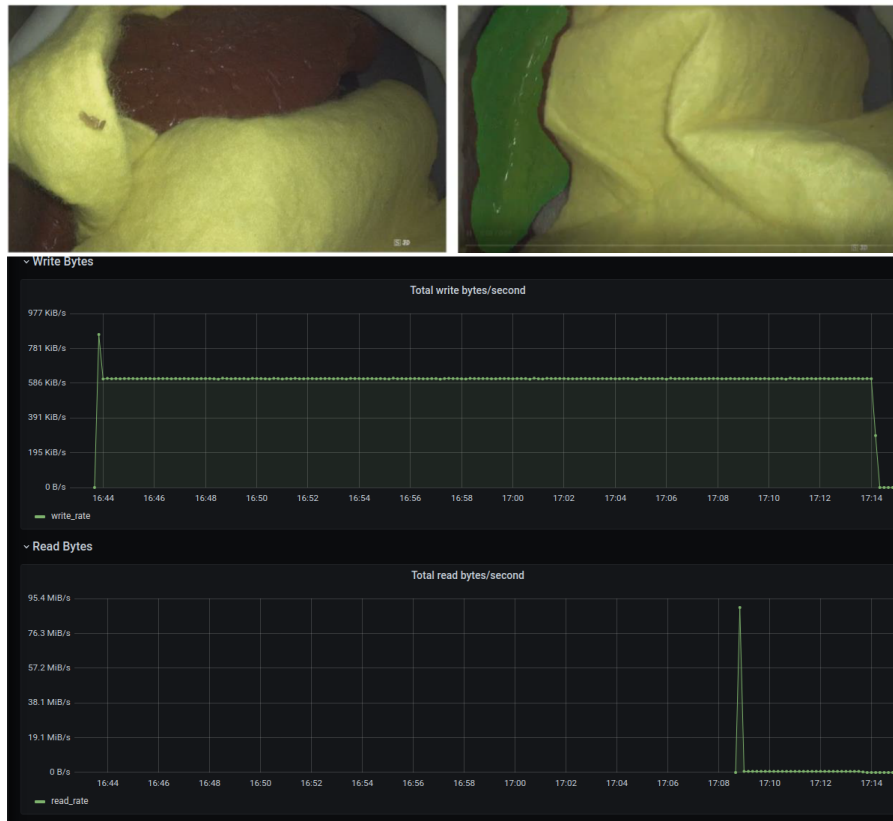


Figure 36: Early results running containerized AI inference jobs (upper figure) and executing a batch reader on a video stream (lower figure).

in which Pravega is ingesting a video stream for 25 minutes. Then, we release a video reader that starts reading the video stream from the beginning. As can be seen, the video reader catches up with the video writer in a couple of seconds, exhibiting a read throughput of 95MBps. This experiment gives a sense that the streaming infrastructure built for NCT can also serve batch analytics jobs.

*ucKPI1: We have tested the execution of containerized AI inference models in our streaming PoC for video analytics. We have also shown how Pravega performs automatic tiering of stream video data, which allows serving batch AI jobs. Overall, our PoC provides a more productive, Cloud-Edge friendly infrastructure for computer-assisted surgery use cases like NCT.*

**Benchmark 1: End-to-end video frame latency measurement via GStreamer.** The goal of this first experiment is to understand the relative impact of Pravega IO latency on a video inference pipeline. Our main focus in this set of experiments are latency metrics: i) *end-to-end latency*, which is the time it takes for a video frame to be durably written to Pravega and read back immediately after, and ii) *inference latency*, which is the time it takes for an AI inference model to process a single video frame. In Fig. 37, we show one of the NCT AI inference jobs used to evaluate the inference latency compared to the IO end-to-end latency. Concretely, we use two inference jobs in this experiment: i) a liver segmentation job, and ii) and surgery instrument identification job. The video inference jobs are deployed on a separate Kubernetes node with access to a GPU. In Fig. 37, we can compare the latency values related to IO and video inference. Visibly, while the distribution of inference latency is not heavy tailed (i.e., GPU has a more stable behavior than network/drive), most video inference latency values are significantly higher than IO latency. For instance, at p90, inference latency is around 2x higher than IO end-to-end latency for local drives (AWS EKS, on-premises cluster). This means that ingesting video data via Pravega is affordable, given all the data durability and management benefits it provides. Interestingly, we observe that running a video inference job without a GPU is virtually

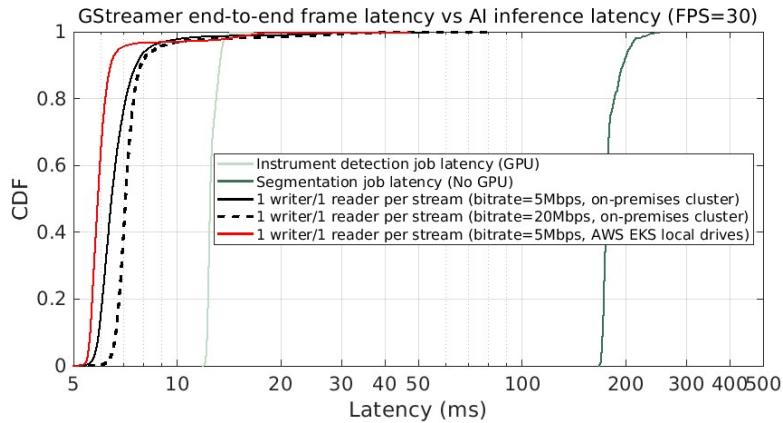


Figure 37: Latency comparison of video frame IO vs. inference latency.

impractical, due to the high latency that may result from running inference on a regular vCPU. This is an interesting hardware aspect to be considered when deploying Pravega at the edge, if a customer needs multiple video inference jobs to be running in parallel.

*ucKPI2, KPI3: With a proper storage layout, the video frame end-to-end latency represents only a fraction of than the actual inference latency in the NCT models tested (e.g., end-to-end latency is 45% lower than inference latency at p95). Therefore, our PoC is achieves good enough performance for real-time AI inference, while providing other key guarantees (e.g., durability, consistency).*

**Benchmark 2: Induce unavailability of long-term storage while ingesting video data.** The infrastructure serving the NCT use case may be encompass edge and cloud resources; i.e., the edge may host Pravega instances ingesting video streams with low latency along with AI inference models, whereas data may be stored in the long term on an external storage service for future batch processing. This yields that the network connection between the edge infrastructure and the long-term storage for video streams may be unreliable. This can be problematic for a tiered storage system for data streams like Pravega, as it was initially designed for a data center environment where a long-term storage service is expected to be (almost) always available. Without getting into too much detail, upon an event write, Pravega temporarily stores that event in the in-memory cache after getting the acknowledgement from the write-ahead log. The data will be sitting in the cache and tagged as “unevictable” until the subsystem that performs data tiering marks the event as safely stored in long-term storage. But, in the case that long-term storage is unavailable, Pravega may only continue ingesting data as long as there is space in the in-memory cache. Beyond that point, Pravega will throttle writers and stop ingesting data. In the case of NCT, this may lead to stop video ingestion and AI inference during a surgery, which is undesirable.

As we describe in deliverable D3.3, for alleviating the impact of long-term storage unavailability in Pravega we have integrated Pravega with IBM GEDS. With this integration, Pravega believes to be storing data on an external storage service, but it is GEDS the one taking care of data tiering on its behalf. By doing this, GEDS can exploit local storage and apply smart tiering algorithm that Pravega does not implement as of today. Fig. 38 shows an experiment that compares the ingestion buffering capacity of Pravega with and without GEDS. In this experiment, Pravega is configured with an in-memory cache size of 1.5GB, whereas the local storage configured in GEDS is 5GB. The ingestion throughput is generated via GStreamer video writers which write at  $\approx 15$ MBps. During the experiment we disconnect the long-term storage service storing stream data (MinIO) to measure the ingestion buffering capacity of the system with no long-term storage available. Visibly, Pravega along can handle workload ingestion for 77 seconds, whereas Pravega with GEDS can last 292 seconds. This represents an improvement of 3.8x in terms of ingestion buffering in front of long-term storage outages. Note that this experiment uses a small GEDS volume; we could consider much larger GEDS

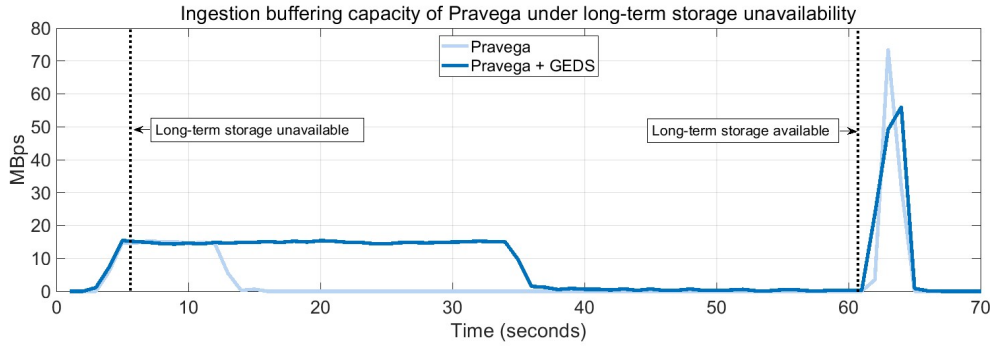


Figure 38: Improvement of Pravega ingestion buffering capabilities when integrated with GEDS.

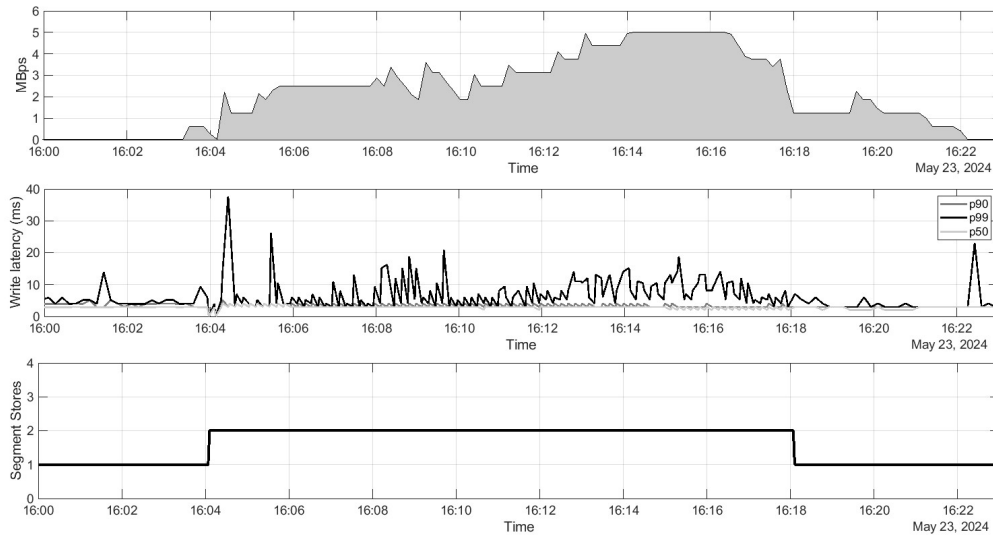


Figure 39: Predictive, trace-based auto-scaling of Pravega instances based on NCT traces.

volumes, as local storage may be a more abundant edge resource compared to memory.

*ucKPI3, KPI8: The Pravega + GEDS integration can improve the tolerance to long-term storage outages in orders of magnitude (e.g., 3.8x in Fig. 38) by exploiting local storage and smart tiering of stream data. This is key for reliably serving NCT video analytics in the Cloud-Edge Continuum.*

**Benchmark 3: Generate fluctuating video streaming workloads and evaluate auto-scaling.** In this set of experiments, we focus on the ability of Pravega to auto-scaling for handling fluctuating workloads. This is vital given the strong usage patterns observed in NCT surgery room occupancy traces. In particular, we focus on the ability of the system to effectively to increase and decrease the number of instances via an external component that may orchestrate Pravega based on some AI/ML techniques. Note that we have explored some predictive AI/ML technique to auto-scaling Pravega, and we report the results in deliverable D5.1.

Fig. 39 shows a trace-based experiment on AWS EKS. We developed a trace-based orchestrator that can instantiate video writer/reader pairs of pods in a Kubernetes cluster based on an input trace from NCT. Similarly, such orchestrator can also change the number of Pravega instances according to an input trace. In this experiment, we executed a 1-day workload trace from NCT (at 20X speed) in which we assumed that every surgery room has a single video writer/reader pair. Moreover, with our predictive auto-scaling algorithm presented in deliverable D5.2, we generated a trace that dictates the number Pravega instances to run. As visible in Fig. 39, the ingestion workload in Pravega



follows a daily pattern in which the central hours of the day exhibit a much higher workload than the nights. In this sense, we can also observe in Fig. 39 that the predictive auto-scaling algorithm triggers the auto-scaling of a new Pravega Segment Store instance before the workload reaches its peak. This demonstrates the ability to auto-scale the Pravega streaming infrastructure to accommodate workload changes. Moreover, in this deployment, we can also observe that the Pravega write latency at p99 obtained from the system metrics falls under 20ms most of the time, which is a great value considering video streams of 30 fps (p90 write latency is  $\approx 4$ ms).

*ucKPI4, KPI7: A streaming storage infrastructure based on Pravega does not only achieve good write latency, but it can also accommodate fluctuating workloads. This is an important advantage for NCT and an interesting substrate for designing auto-scaling policies in the Learning Plane.*

**Experiment 2: Confidential execution of NCT AI models (ucKPI5, KPI6).** In this experiment, we automate a legacy application translation into a confidential TEE-enabled application. TUD and NCT are working together to create an image where the liver segmentation processing could be done in a confidential way. At the time of this writing, we have successfully translated the native image to an Intel SGX-compatible image with zero development cost, requiring only changes to the base Dockerfile. As for other components such as ROS core, ROS replaying program, and ROS image viewer, are not part of this effort.

In Fig. 40, we show “liver segmentation” with confidential execution guarantees (bottom-right). Other parts of the figure shows other components such as ROS core (top-right), image viewer (top-left), and message measurement (bottom-left). At the moment, those components are not protected by Intel SGX. Very succinctly, we have run a preliminary experiment to compare native execution with SCONE hardware mode (executed inside an Intel SGX enclave) and SCONE simulation mode (executed only in SCONE that simulates Enclave behaviour). Our preliminary results show that the hardware and simulation mode introduce overhead as much as  $21.6\times$  and  $3.6\times$  compared to native execution. We suspect this is caused by the large memory usage that cannot be handled by Intel SGX. Improving the performance is already a work in progress.

```

root@sgx15:~# source /opt/ros/noetic/setup.bash
root@sgx15:~# roslaunch image_view video_recorder image:=/image_overlayed
[ INFO] [1718111762.118569695]: Waiting for topic /image_overlayed...
[ INFO] [1718111770.540405723]: Starting to record MJPG video at [960 x 540]@15fps. Press Ctrl+C to stop recording.
[ INFO] [1718111849.410568695]: Recording frame 3

started roslaunch server http://sgx15:39203/
ros_comm version 1.16.0

SUMMARY
=====

PARAMETERS
* /roscpp: noetic
* /rosversion: 1.16.0

NODES

auto-starting new master
process[master]: started with pid [6023]
ROS_MASTER_URI=http://sgx15:11311/

setting /run_id to c38d7f5c-27f1-11ef-8058-ac1f6b...
process[rosout-1]: started with pid [6034]
started core service [/rosout]

no new messages
no new messages
no new messages
no new messages
average rate: 0.117
  min: 8.260s max: 9.603s std dev: 0.37058s window: 11
no new messages
no new messages
no new messages
no new messages
no new messages
no new messages
no new messages
no new messages
no new messages
average rate: 0.117
  min: 8.260s max: 9.603s std dev: 0.35506s window: 12
no new messages
no new messages
no new messages
no new messages
no new messages

[54/158] [SCONE] Using the following values (default values)
SCONE_QUEUE=4
SCONE_SLOTS=256
SCONE_SIGPIPE=0
SCONE_MMAP32BIT=0
SCONE_SSPTS=100
SCONE_SSLEEP=4000
SCONE_CONFIG=/etc/sgx-musl.conf
SCONE_TCS=8
SCONE_LOG=WARNING
SCONE_HEAP=4294967296
SCONE_STACK=2097152
SCONE_ESPINS=10000
SCONE_MODE=hw
SCONE_ALLOW_DLOPEN=yes (unprotected)
SCONE_MPROTECT=no
SCONE_FORK=no
SCONE_FORK_OS=0
SCONE_CONFIG_ID: <not specified>

```

Figure 40: Showcase of running a liver segmentation model inside an Intel SGX enclave.

*ucKPI5, KPI6: A zero development effort approach for translating existing container image to TEE-compatible image has been done. This method could be applied to another use case within CloudSkin project. Current performance overhead has been observed and will be addressed as soon as possible. Using Intel SGX enclave should protect both the computation as well as patient data as mentioned in the KPI.*

## 5.4 Use Case: Agriculture

### 5.4.1 Overview

This use case addresses the problem of sharing agricultural data. Most sensor management solutions are closed platforms from manufacturers or sensor installation and analysis solution providers, and data is centralized on private servers. Furthermore, there is great reluctance to transfer data due to loss of control over its use and fear that the data will be used by competitors.

This represents a barrier to the creation of an ecosystem of companies, applications, and services that can exploit agricultural information by making data available to different interested agents, not only governments and environmental inspection agencies, but also intermediaries and providers of other information services.

The provision of an environment that facilitates the exchange of information allows farmers and society in general to benefit from greater productivity and the emergence of new related business services, supported by the application of information technologies.

The **agricultural use case** proposes to fulfill the above needs through two main experiments:

- **Experiment 1: Agricultural Dataspace**

There are several actors interested in the use and exploitation of agricultural data, although their interests are very different. While farmers want to monitor their land and optimize their production, the government wants to be able to inspect their environmental impact. Moreover, data brokers want to sell data to potential customers, and other companies and entities may seek to create solutions or analytical studies, for scientific or technological purposes.

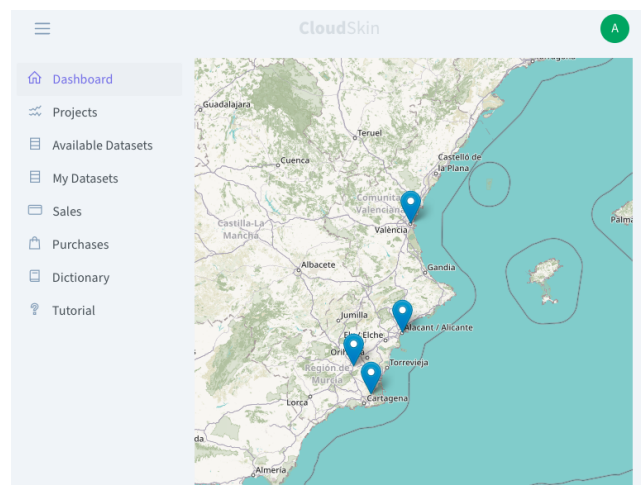


Figure 41: Agricultural Dataspace main menu.

To unite the different interests of data access, the developed functional PoC of a dataspace (see Fig. 41) establishes a system of contracts, which allows compliance with the legal restrictions and interests of data providers and consumers, and at the same time allows them to monetize their information, thus encouraging their updating of agricultural data sources.

It is not enough for the information to be accessible, but for being useful its meaning, it must be understood. To solve the problem of different types of measurement units and meaning of data, the Dataspace establishes a common data dictionary (see Fig. 42), which gives meaning to the shared information. The information is accessible through a data source search engine, which allows you to subscribe or acquire the information (see Fig. 44).

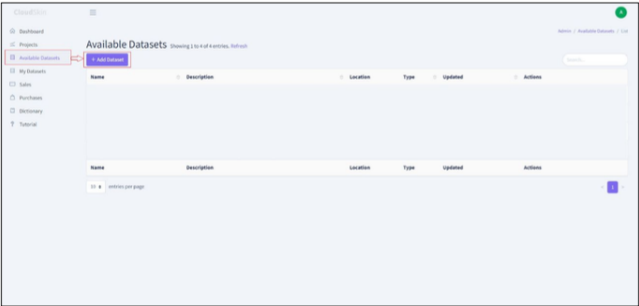
Of course, we do not leave aside the main objective of the Dataspace, which is to share data. To do this, the Dataspace implements a data upload system and generates a link to share it with

interested users, proposing a mechanism to standardize data communication between different platforms, regardless of the origin of the data or the sensor manufacturer.



Data type	Default unit
PM10 media	%
PM2.5 media	%
SO2	%
NO2	%
Milisiemens por centímetro	uS/cm
Porcentaje	%
Pascales	kPa
Long Date	YYYY-MM-DD H:i:s
Temperature °C	°C
Grados	00,0

Figure 42: Data dictionary.



Name	Description	Location	Type	Updated	Actions
------	-------------	----------	------	---------	---------

Figure 43: Dataset creation.

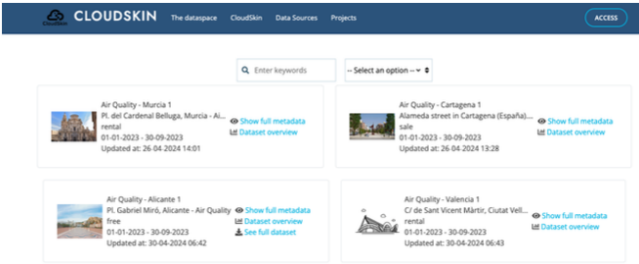


Figure 44: Public search of datasources.

Additionally, and taking advantage of the experience of similar projects related to dataspace, such as the METASPACE2020 . eu in the project, the planned functionality has been expanded with the inclusion of the “Data Project” concept. This concept provides an added value by making projects that consume data visible, allowing them within the project to not only to obtain the information, but also to process it and automatically update or insert a new data source related to the project. This functionality lays the basis for the construction of information processing layers, while promoting the expansion of the agricultural Dataspace for both scientific and commercial uses and of interest to society.

**Main components.** Here we provide a brief summary of the main components of the Dataspace:

**Datasets.** Datasets form the core of the application and store detailed information about the data sets available in the system. Each record in this table contains key attributes such as dataset owner, origin, price, access type (sale, rental, free), and additional metadata such as description, category, and creation date.

Fig. 43 to Fig. 47 shows how a user can create a dataset. This includes basic metadata such as the dataset name, the dataset owner, its origin, etc. It is also possible to define the way in which the data will be shared, whether “rent”, “sale” or “free” (see Fig. 45). A dataset owner can also indicate the exact area in which the data has been taken, which can include the latitude and longitude of the area. Later, a pin will be displayed on a map, showing the area where the data was collected.

The screenshot shows the first part of a dataset creation form. It contains several input fields and a dropdown menu. The fields are: 'Dataset name' (filled with 'Air Quality - Murcia 1'), 'Owner name' (filled with 'Albermarco'), 'Origin' (filled with 'Pl. del Cardenal Belluga, Murcia'), 'Start Date Range' (filled with '01/01/2023'), 'End Date Range' (filled with '30/09/2023'), 'Sales method' (dropdown menu with 'Rental' selected), 'Selling price (€)' (filled with '15'), 'Dataset description' (filled with 'Pl. del Cardenal Belluga, Murcia - Air Quality'), and 'Image' (filled with a file path 'xLOkV5gghC1ceY9WwLSRR2eW7D0A0N5jd2K.jpg'). There is a 'Browse' button next to the image field. A note below the price field states: 'The sales price for rental cases will be on a monthly basis.'

Figure 45: Dataset creation form 1/3.

The reason for creating the dataset is the availability of the data within this dataset. In order to upload the information we will have a section in the form: “Format of generated data”, where the structure of the data can be defined. Static data will be loaded, it is essential to keep in

☒ Geo-referenced data ?  
If the data are geo-referenced, when sending data via API it will be necessary to indicate the latitude and longitude of the point of interest, using the "latitude" and "longitude" fields.

Latitude  Longitude

License

Category

Data use contract

Upload an optional data use contract, if you do, you will have to validate manually that the buyer has signed the contract.

☐ Auto-validate sales  
Automatically validates sales if this field is checked.

Figure 46: Dataset creation form 2/3.

mind that the names of the fields must be defined in the same way and order in the loading Excel (XLSX and XLS supported formats) and in this case you must choose in the "DataType Upload" the "Static Data" option. The reason for creating the dataset is the availability of the data within this dataset. In order to upload the information we will have a section in the form: "Format of generated data", where the structure of the data can be defined.

Format of generated data

Field name	Data type	Unit	Description
Time	Select Field		
PM10 media	PM10 media	%	Partículas PM10
PM2.5 media	PM2.5 media	%	Partículas PM2.5

If you need to create any additional data types, please contact the administrator at [admin@alternateno.es](mailto:admin@alternateno.es)

Data Type Upload

Static Data

File Data

Figure 47: Dataset creation form 3/3.

Once imported, the data is stored in the DB "datareads" (see Fig. 48). Additionally, the data is visible within the platform (see Fig. 49):

- Engines

The engines play a fundamental role, since they contain the basic information that will contain the data that will be imported into the datasets. As we mentioned in the previous point, this structure is defined as "Format of generated data" (see Fig. 50).

- Datareads

Datareads are records that contain a set of structured data, given by the engine, with valuable information for the dataset. We can loop up these datareads within the dataset, in the "Last 5 records" section (see Fig. 51):

	id	dataset_id	latitude	longitude	data	created_at	updated_at
1	100001	100001	40.4168	-3.7038	"Time": "10:00", "PM10 media": "100", "PM2.5 media": "100"	2024-06-28 11:29:08	2024-06-28 11:29:08
2	100002	100001	40.4168	-3.7038	"Time": "10:05", "PM10 media": "100", "PM2.5 media": "100"	2024-06-28 11:29:08	2024-06-28 11:29:08
3	100003	100001	40.4168	-3.7038	"Time": "10:10", "PM10 media": "100", "PM2.5 media": "100"	2024-06-28 11:29:08	2024-06-28 11:29:08

Figure 48: DB Datareads.

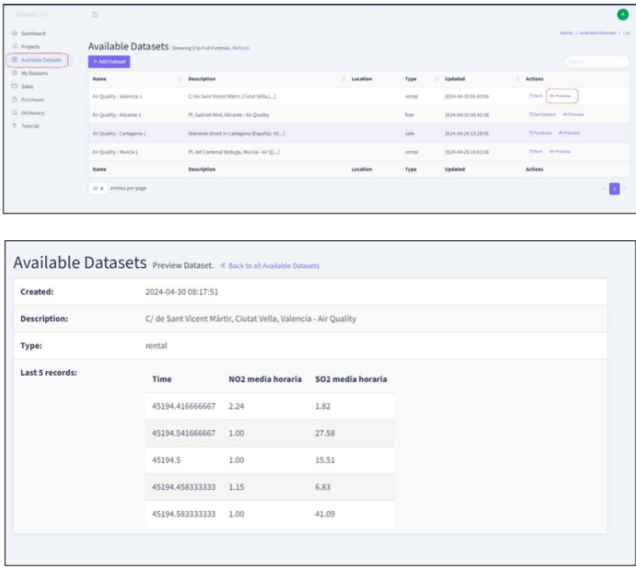


Figure 49: Available dataset information.

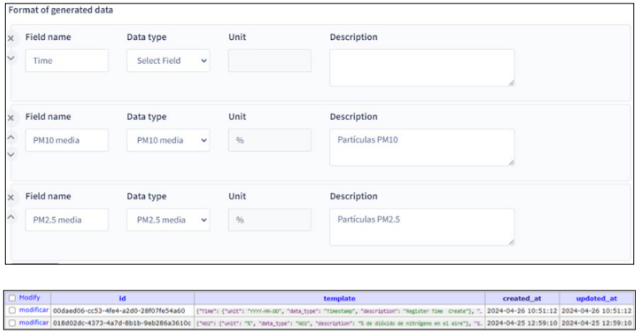


Figure 50: Data format definition.

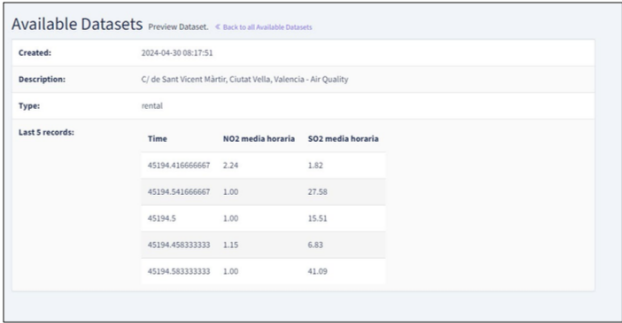
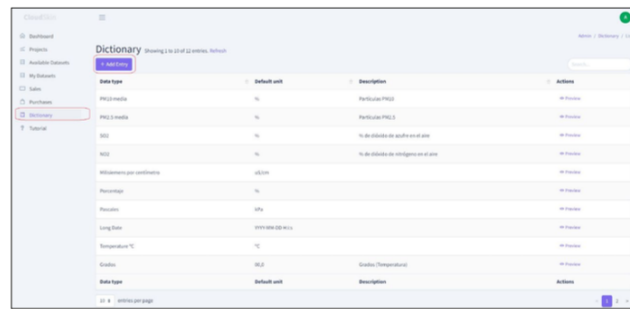
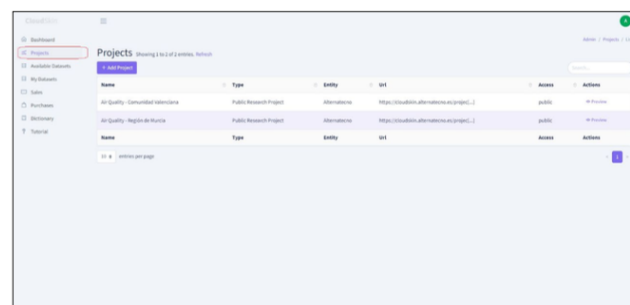


Figure 51: Last 5 records screenshot.



Data type	Default unit	Description	Actions
PM2.5 media	%	Partículas PM2.5	Alt. Proceso
PM2.5 media	%	Partículas PM2.5	Alt. Proceso
NO2	%	% de dióxido de nitrógeno en el aire	Alt. Proceso
NO2	%	% de dióxido de nitrógeno en el aire	Alt. Proceso
Mediciones por coordenada	ubicación		Alt. Proceso
Presión	%		Alt. Proceso
Presión	hPa		Alt. Proceso
Longitud	latitud/longitud (WGS)		Alt. Proceso
Temperatura °C	°C		Alt. Proceso
Grados	NO2	Grados (Temperatura)	Alt. Proceso
Data type	Default unit	Description	Actions

Figure 52: Data dictionary.



Name	Type	Entity	URL	Actions
Air Quality - Comunidad Valenciana	Public Research Project	Alternativa	<a href="https://cloudskin.uberinteligencia.es/project_2">https://cloudskin.uberinteligencia.es/project_2</a>	publico
Air Quality - Región de Murcia	Public Research Project	Alternativa	<a href="https://cloudskin.uberinteligencia.es/project_2">https://cloudskin.uberinteligencia.es/project_2</a>	publico
Name	Type	Entity	URL	Actions

Figure 53: Project form.

## • Dictionary

The dictionary entity stores basic information related to the data used to describe data types, default units, and description of the data that we are going to import into the datasets. This component facilitates the definition and structure of the data that the dataset datareads will contain. To create a new one, you will need to be a system administrator.

The data shown in Fig. 52 will be the data types within the dataset engines that we create, which we will have available, making it possible for the administrator to create new data types.

## • Projects

The projects are associated with the use of a set of datasets, which make up a logical data entity. In order to create a “Project” it will be necessary to have previously created datasets. The information that the “Project” will contain will be composed of basic data such as the name, owner (Entity), the type of access to the data, and of course, the datasets that will be part of it (see Fig. 53).

## • Purchases and sales

Purchases contain information about the acquisition of datasets by users. In order to acquire a Dataset, it will be accessed from the “Available Datasets” (see Fig. 54).

Once the purchase is requested, the owner of the data set may accept or deny the acquisition within the conditions of use determined by the data provider. The owner will be able to view all the transactions pending management from the Sales menu.

If the owner finally accepts the purchase, the Purchases form will display the acquired datasets.

## User management

User, role, and permission tables are essential for user authentication and authorization within the Dataspace. Roles and permissions control access and operations allowed in the system, ensuring data security and integrity.





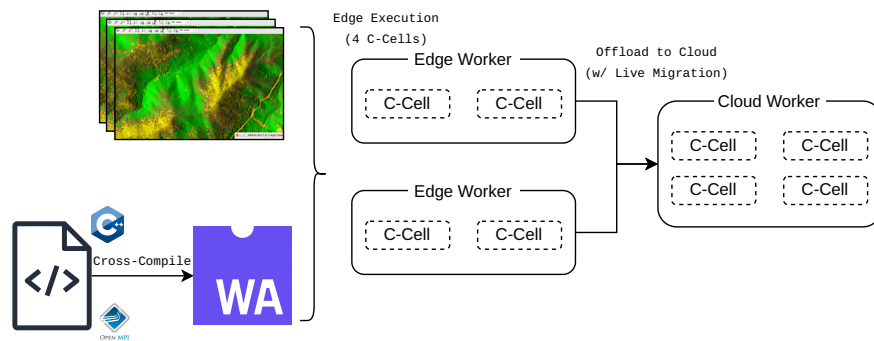


Figure 56: Vegetation Index Calculation from Geo-Spatial Images with C-Cells.

size of the batch of images increases.

Fig. 56 illustrates this process. First, C/C++ code written using the GRASS and OpenMPI API is cross-compiled to WebAssembly. As a result of this cross-compilation, and as described in D4.1, some symbols are left un-resolved like, for example, the OpenMPI API. Their definition will be provided by Granny at runtime, in order to implement transparent live-migration. Some other symbols, like those corresponding to the GRASS API, are resolved at compilation time because all transitive dependencies of GRASS are also cross-compiled to WebAssembly and statically linked. The WASM code is then uploaded to the CloudSkin platform, and, when geo-spatial images arrive, will start processing them. At the beginning, to minimize data movement, this processing may happen in lower-powered edge nodes, geographically distant from each other. After a while, when computational demands increase, the C-Cells executing the code can be live migrated to a powerful cloud server to improve locality of execution and reduce execution time. In D4.2 we include experiments that quantify the improvements in execution time.

#### 5.4.2 Status of the use case at M18

On the one hand, we have developed a dataspace, which allows the management and exploitation of data sources by the different interested roles.

Agricultural management is complex to analyze due to the extreme diversity, derived from the biological and environmental variety, given the diversity of tasks and challenges in agriculture.

At M18, a first version, very close to a market product, of an agricultural Dataspace has been deployed. This platform contemplates the key aspects to consider for its management, including the semantics of the data through the use of a data dictionary, and the contractual complexity derived from information management needs.

Actually, a massive collection of data from agricultural sensors is being carried out, which will be joined as part of the experiment to the Dataspace datasets, allowing the behavior of the platform to be analyzed with real data loading experiments, and the benefits of optimized data management, via dynamic resource management in the continuum. This dataspace, already implemented, will allow the analysis of the performance derived from the exploitation of the data sets.

On the other hand, the integration with C-Cells and C-Cell migration using the Granny system is a work-in-progress. The geospatial software stack is large and includes many transitive dependencies as static libraries. Every single one of this dependencies needs to be cross-compiled to WebAssembly too, and this is a sometimes arduous process. Once the cross-compilation process is finished, we should be able to transparently migrate geospatial MPI applications from the edge to the cloud (or the other way around) in response to changes in demand.

### 5.4.3 Why this use case needs the compute continuum?

A dataspace, understood as a software platform for the controlled exchange of data between users and services, including direct reception from sensors, can benefit from a continuum infrastructure.

While the agricultural Dataspace focuses on the study of the complexity of data use and sharing, it can serve as a use case for the analysis of continuous computing services. Due to the large volume of data that may require processing in the processes of loading, sharing and managing data in the Dataspace, you will directly benefit from the possibility of dynamic allocation of resources and services in the continuum, actively exploiting its adaptive advantages, for instance, by keeping data at the edge data servers but exfiltrating some CPU-bound tasks to the cloud.

Lastly, the calculation of vegetation indexes using C-Cells and MPI necessitates the cloud- edge computing to navigate the trade-off between data locality and compute resources' availability. First, when there are not many images to process, it makes sense to run the MPI application where the images are located. However, the parallelism available at the data source may be limited, and a bottleneck when we have more images to process. At this point it may be adequate to live migrate the running MPI processes to a cloud-grade machine with more available parallelism.

### 5.4.4 Where AI helps in this use case?

In the case of adaptation to the continuum, AI can be used for the dynamic selection and management of resources necessary for the processing of data in the Dataspace.

Additionally, and in a more global way, AI could be applied to the analysis and exploitation of information shared by an agricultural and environmental Dataspace in multiple ways. For instance, AI can be used to optimize scarce resources such as water or electricity consumption, using machine learning algorithms to achieve sustainable crop management.

In terms of the continuum, as in the rest of use cases, AI can be used to optimize two main aspects: 1. **Task placement**, i.e., decide where (edge or cloud) to process the agriculture processing tasks (e.g., sensor data processing, NDVI index calculation, radiation calculation, etc.), given the availability of computing resources at the edge; and 2. **Scaling**, i.e., decide upon horizontally scaling the resources at the edge with on-demand cloud resources to support all the allocated tasks.

### 5.4.5 Experiments, KPIs and benchmarks

The specific use case KPIs (ucKPIs) are defined in Table 15.

Table 15: Summary of use case-specific KPIs for agriculture.

ucKPI	Description
ucKPI1:Apdex Score	The Apdex score of the main web pages of the Dataspace should be greater than 0.85 (good).
ucKPI2: Time to First Byte (TTFB)	The time required to request information from the server and to transfer the information that was requested should be below 1 second.
ucKPI3: C-Cell Execution Overhead	Executing geo-spatial software running MPI with C-Cells must have negligible performance overheads.
ucKPI4: Cloud-Continuum Execution Overhead	Live-migrating geo-spatial software across the cloud continuum must have limited overhead, and should always improve end-to-end execution time compared to not migrating at all, and re-executing.

To validate our contributions, we propose the following experiments:

- **Experiment 1. Validation of the agricultural Dataspace.** The validation of the agricultural Dataspace has been carried out objectively, as an experiment of functionality, usefulness and performance for the use of the data. This experiment includes loading experiments and creating datasets, performing performance analysis and loading functionality with both static data and data coming from an external service.

- **Experiment 2. Continuum integration analysis.** The major goal of this experiment is to run a vegetation index software using geo-spatial images and MPI using C-Cells, and live migrate parts of the application from one machine to another one.

#### 5.4.6 Early results

**Experiment 1. Validation of the agricultural Dataspace.** A PoC has been released that facilitates the negotiation and understanding of the data, establishing a strong basis for the application of AI and development of services applied to agriculture and environmental analysis. The PoC runs within a container instance at the KIO Networks edge datacenter.

By now, the validation of the Dataspace is still work-in-progress. As a first key metric, we plan to measure user satisfaction using the **Apdex<sup>12</sup> score**. This score is an industry standard to measure the performance of enterprise applications. The Apdex method converts many measurements into one number on a uniform scale of 0 to 1, namely 0 = **no users satisfied**, 1 = **all users satisfied**. This metric can be leveraged to report on any source of end-user performance measurements for which a performance objective has been defined. The key idea is to use this score by stating a goal for how long a specific application transaction or request must take. Those transactions are then labeled as failed, too slow, tolerating (sluggish), or satisfied (fast) requests.

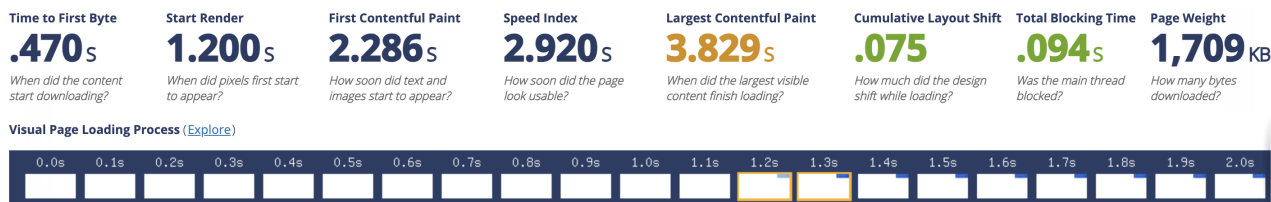


Figure 57: Dataspace performance metrics.

The metrics in Fig. 57 provide insight into the initial performance characteristics of the Dataspace, considering a low-load production environment and a single container. This information serves as an online basis for analyzing Dataspace access performance before and after applying performance improvement mechanisms.

Early results reflect that the Dataspace is fast to connect and deliver initial code, although it began rendering content with little delay. Initial tests do not detect security problems.

Furthermore, we will use other metrics such as the **time to first byte (TTFB)**. The TTFB is the time required to request information from the server and to transfer the information that was requested. In simple terms, it is the time from the point where you navigate to a web page through to when it starts to render. This period of time includes the server request, which can differ according to internet connection and location; the time needed to process a request or form a response; and finally the time needed to send the response back to the client.

Return time equals 40% of the total TTFB. The slower the TTFB, the more time it will take for your user to view any content on your site.

The waterfall diagrams in Fig. 58 and Fig. 59 demonstrate low TTFB and good performance of the Dataspace. The greatest delays are the loading of external content, derived from components of the backpack.

**Experiment 2. Continuum integration analysis.** In this second experiment we run the GRASS geo-spatial software suite to calculate vegetation indexes on satellite images using C-Cells executing MPI code. The first part of the experiment studies the baseline overheads of executing the application using C-Cells instead of MPI processes.

<sup>12</sup><https://www.apdex.org/>

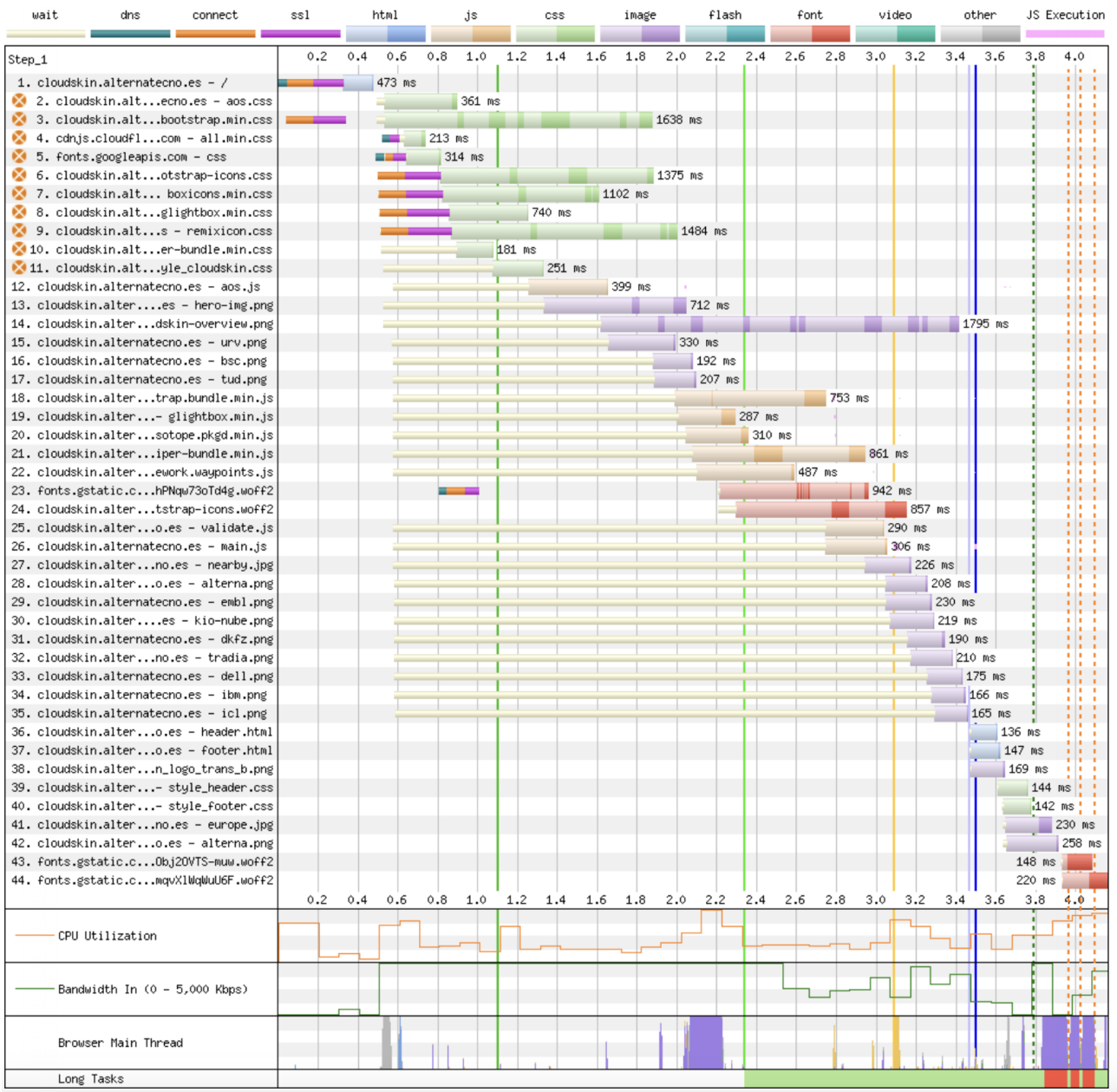


Figure 58: Waterfall view.







	Waterfall	Screenshot
<div>First View (3.493s) <a href="#">Timeline (view)</a> <a href="#">Processing Breakdown</a> <a href="#">Trace (view)</a></div>		
Run 2:		
<div>First View (3.632s) <a href="#">Timeline (view)</a> <a href="#">Processing Breakdown</a> <a href="#">Trace (view)</a></div>		
Run 3:		
<div>First View (3.551s) <a href="#">Timeline (view)</a> <a href="#">Processing Breakdown</a> <a href="#">Trace (view)</a></div>		

Figure 59: Dataspace waterfall initial tests.

## 6 Description of testbeds implementation and setup

To realise the use cases in the project, it is essential to establish the continuum infrastructure. Non-surprisingly, this involves the deployment of various software components over the corresponding continuum resources. This section provides a detailed description of these processes for each of the use cases.

### 6.1 Testbed for the mobility use case

The testbed is located in Castellolí Parcmotor Circuit. More specifically, for the CloudSkin project and mobility use cases, cloud-edge hardware have been defined:

- **Cloud (Control Room):** Two virtual machines have been deployed in Lenovo SR650 servers running VMware vSphere to create different virtual machines (VMs) for various services and apps (referred to as the “Local Cloud”) with the purpose of running services in “the cloud.”
- **Edge (Pole):** Node 1, A Samsung Wisenet PNO-9080R camera captures real-time images from a specific area of the circuit and sends these images to other devices via the RTSP protocol for analysis. Additionally, an SE350 edge server has been deployed in Node 1 with the purpose of running services at “the edge.” Node 1 also features an energy control and management system called ORION.

For the software stack in the testbed, we mainly use NearbyOne orchestrator platform, which is responsible for the service onboarding and life-cycle management (LCM) of cloud-native apps and infrastructure at a global scale, across the edge-to-cloud continuum. NearbyOne Orchestrator plays the role of the multi-cluster orchestration engine in CloudSkin. The NearbyOne solution leveraged in CloudSkin is mainly composed of:

- The **NearbyOne Orchestration Platform**, the main component of the solution, is in charge of performing all tasks related to the orchestration of applications and infrastructure.
- The **Nearby Blocks** are distributed components that encapsulate logic and code for different application-specific functionalities.
- The **NearbyOne observability stack**, built upon cloud-native open-source technologies, and designed to efficiently collect, transport, and aggregate telemetry information from the underlying infrastructure.
- The **NearbyOne Northbound Interface** (NBI) Orchestration API, to enable the communication with the Learning Plane .

Overall, the architecture is shown in Fig. 60. CNX provides the edge and cloud infrastructures and the camera, and NearbyOne provides mechanisms to automate and orchestrate the infrastructure located in Castellolí, and the deployment of components, such as the monitoring stack, the learning plane, or the mobility use case applications. In particular, NearbyOne will manage the set of sites, i.e., the SE350 edge server and the two virtual machines in the private cloud (control room). Each site has been provisioned with a Kubernetes system to manage containerized applications. The NearbyOne controller itself has been deployed in a public cloud, in Amazon Web Services (AWS).



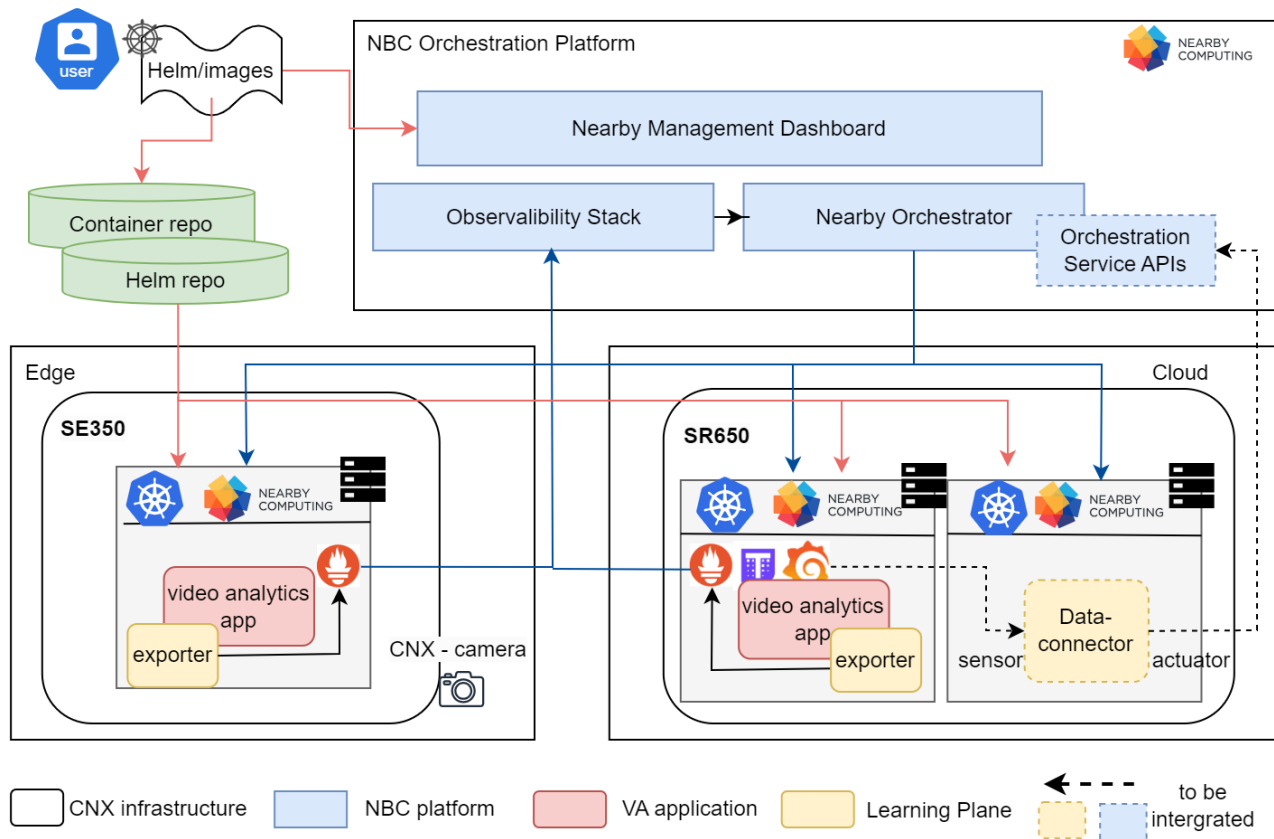


Figure 60: Testbed and deployment architecture for mobility usecase.

## 6.2 Testbed for the metabolomics use case

The METASPACE orchestrator in production runs on an AWS EC2 r6a.2xlarge instance, with 8 vCPU and 64 GB of memory. The system uses the Amazon S3 object storage service to store the annotated molecules as .png images and intermediate results. Final annotation results are saved to PostgreSQL and AWS Elasticsearch for indexing and rapid searching.

Regarding the off-sample service, there is one daemon running on the METASPACE orchestrator, referred to as “update” daemon, which handles the off-sample classification of the incoming datasets. In this case, a dataset is nothing but a collection of .png images outputted in the annotation step of the pipeline. Datasets are enqueued into a message queue and are consumed by four off-sample threads. Each thread groups the images of the dataset into batches of 32 images and posts each batch to the AWS ECS service via an HTTP endpoint. To not overload the inference containers, every off-sample thread manages up to 8 concurrent synchronous batches. Inference containers have 1 vCPU and 2GB RAM. Fig. 61 illustrates the pre-project off-sample service implementation.

To scale out to large datasets, the AWS ECS abides by the following **auto-scaling policy**:

- By default, there is one ECS container up at all times.
- If total CPU usage exceeds 80% for more than two minutes, the auto-scaler adds four additional containers. This scaling keeps going until the maximum of nine containers is reached.
- Conversely, if CPU usage drops below 40%, two containers are recycled.



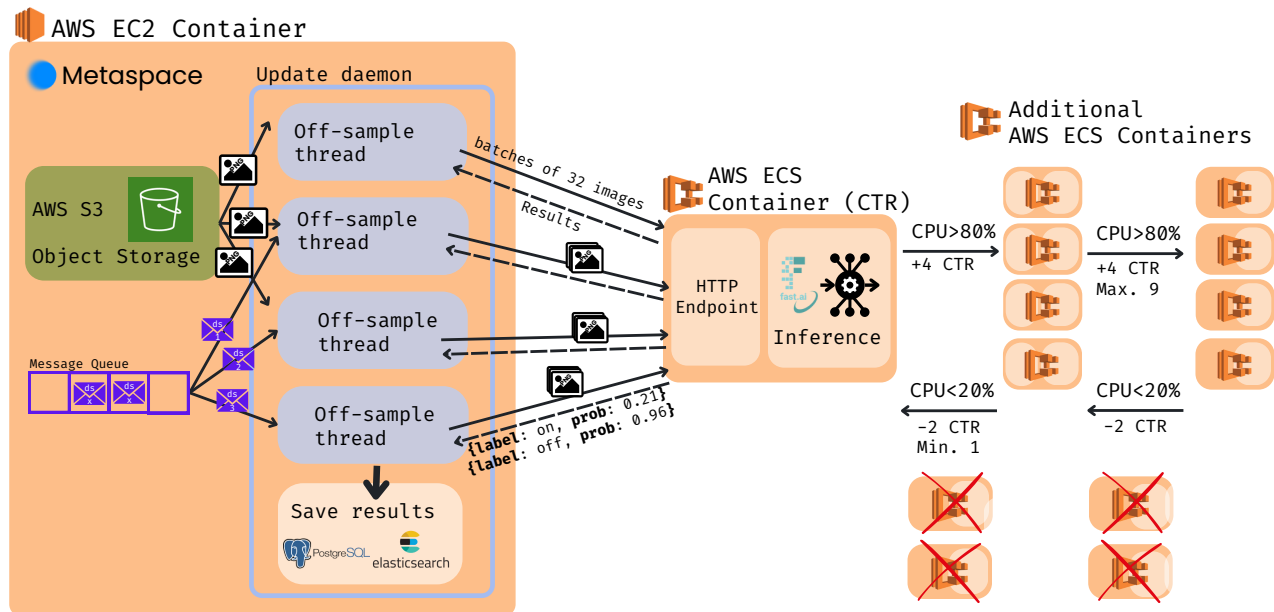


Figure 61: METASPACE off-sample architecture.

For **Lithops Serve**, we prepare two testbeds: 1.- A **cloud-only** testbed, and **edge-only** testbed built upon K8s. The specs of each testbed are the following:

**Cloud-only testbed:** This testbed comprises one AWS EC2 t2.micro instance (1 vCPU; 1GB of RAM) to host the Lithops Serve orchestrator (Batch Manager, Resource Provisioner, etc.) and AWS Lambda functions provisioned with 2 vCPUs and 3 GB of memory each to operate as the executor instances. Also, the testbed uses AWS S3 for function communication, logging and the storage of results.

Jobs can be configured by adjusting:

- **Number of executors:** The number of serverless functions in the pool. For our early results, the number of serverless functions was upper bounded by a maximum of 900 concurrent functions.
- **Batch size:** The number of images per batch, set to 32 by default.

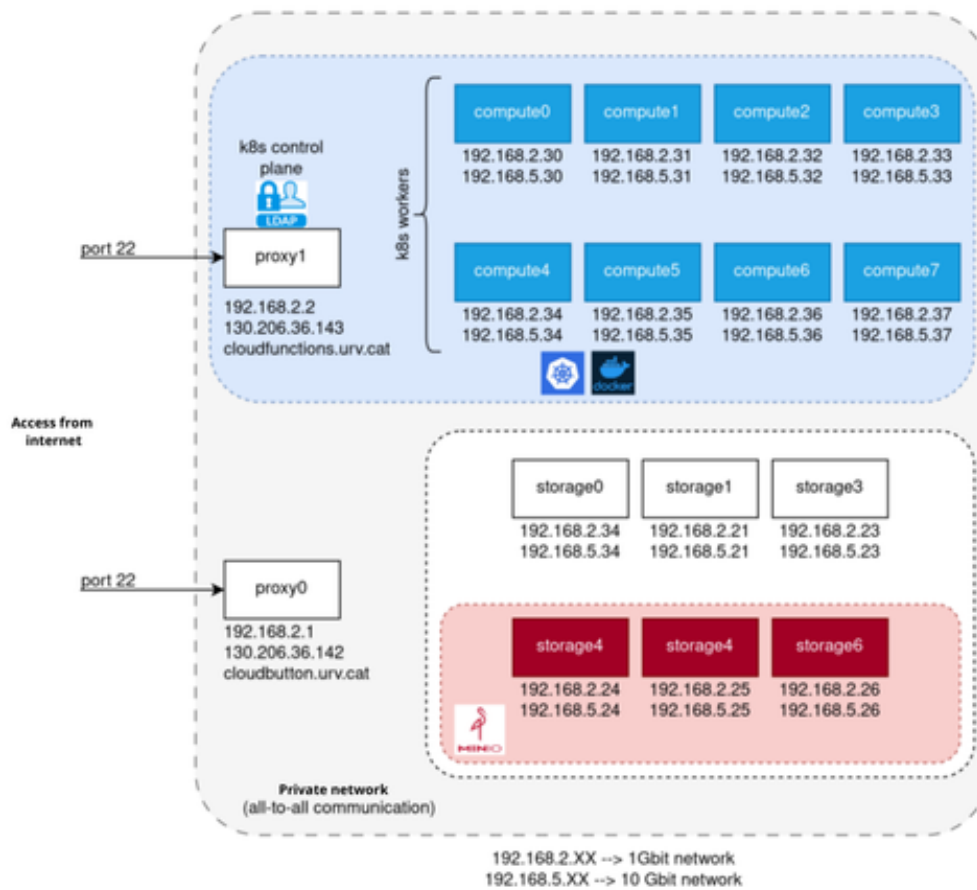


Figure 62: Kubernetes (K8s) testbed at URV for Lithops Serve.

**Edge-only testbed:** This testbed uses an on-premises K8s cluster from URV that comprises a total of 456 CPUs and 548 GB of memory. As the executor instances, Lithops Serve launches pods of 2vCPUs and 3GB memory to be equivalent to the above AWS Lambda executors. As storage backend, this testbed installs MinIO object store, which is very convenient since it shares the same API of AWS S3. Fig. 62 shows the K8s edge tested.

**SCONE testbed:** to test the early stages of confidential computing on Lithops Serve, a local minikube Kubernetes cluster was created on a single machine. The machine has an Intel Xeon CPU E-2186G @ 3.80GHz with 6 Cores, with the Intel SGX architecture extension, and 30 GB RAM. For the preliminary experiment, each Kubernetes pod needs at least 6GB of SCONE HEAP to be created. This means that a maximum of 4 executors can and 1 master be used for now. The Docker Image used for the K8s testbed gone through SCONE framework to be compatible with Intel SGX. The experiment was conducted in 3 types of execution as the following:

- **scone-nofork:** SCONE enables the execution to run in a hardware mode. Meaning it runs on the actual Intel SGX enclave hardware. Here, the experimental forking feature is disabled.
- **scone-dbgfork:** SCONE enables the execution to run in a hardware mode. It also runs on the actual Intel SGX enclave hardware. Here, the experimental forking feature is enabled. Due to this reason, log level was also needed to be set to DEBUG. Theoretically, this reduces performance due to more verbose logging.
- **sconesim-fork:** SCONE runs the test in a simulation mode. This means that it does not utilize the hardware enclave, only on top of the SCONE runtime with simulated Enclave behaviour. This is useful to debug a performance degradation caused by the process of converting a Docker

image to an SCONE-compatible one. Most of the time, the performance in simulation mode can be regarded as the best possible performance for such image.

### 6.3 Testbed for the CAS use case

In terms of testbed infrastructure (see Fig. 63), the cluster used in the NCT use case PoC is built with Dell PowerEdge R750 servers with Intel Xeon CPUs using VMWare VMs. Each VM was given eight vCPU cores, 16GB of RAM, and 250GB of local server NVMe SSD storage. A 100TB NAS server was also used to provide NFS remote storage for performance evaluation. A set of 4 VMs were connected using Kubernetes, with Pravega deployed within the cluster.

Moreover, to compare against a cloud infrastructure, we have also executed video streaming experiments on AWS Elastic Kubernetes Service (EKS). Specifically, the EKS cluster is formed by 3 EC2 i3en.2xlarge instances (acting as Kubernetes nodes) with 2 local NVMe drives each. We used the Rancher Local Volume provisioner [24] plus some scripts to provision local volumes for Bookies, which are the component in charge of the write-ahead log for Pravega.

In both cases, we deploy Pravega as follows: 1 Zookeeper instance, 1 Pravega Controller instance, 1 Pravega Segment Store instance, and 3 Bookkeeper instances. Pravega is configured to use 1 data replica in Bookkeeper per event stored, which we is good enough for video analytics use cases. We use for comparison the latency of NCT AI inference jobs w/wo a GPU.

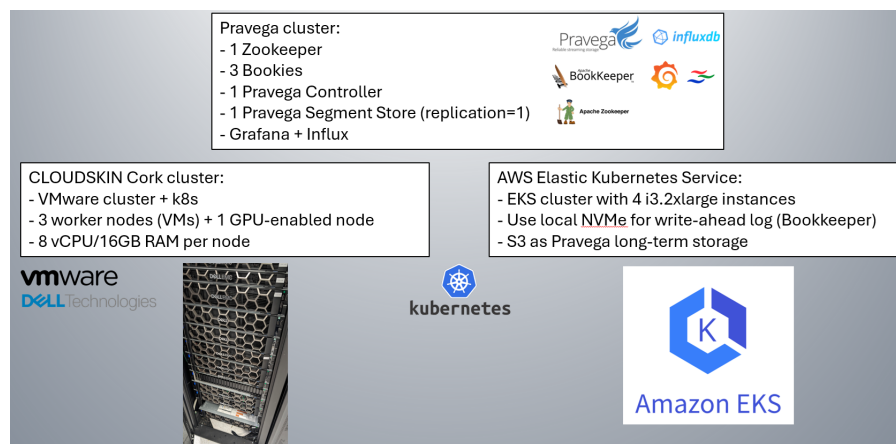


Figure 63: Testbeds provided to support the CAS use case.

### 6.4 Testbed for the agriculture use case

The testbed for this use case has been deployed at KIO Networks. This partner has a data center on which it deploys virtual data center solutions.

More specifically, the underlying hardware that will allow the experiments on the infrastructure defined in Fig. 13 are Cisco servers - UCSB-B200-M4 - UCS B200 M4 Blade Server, racked in a server chassis. The virtualization platform used is VMware ESXi, 7.0.3, 23307199, and for the initial scope of testing, logical restrictions of 30 Mbps bandwidth, 100 Gb and 2 vCPUs have been established.

To enable dynamic resource management and analysis, the platform is deployed on Kubernetes, and includes C-Cells as a means for executing legacy MPI code and migrating tasks between the edge and the cloud.

## 7 Conclusions

This document has detailed the architecture of the CloudSkin platform as well as the interplay among its software components to culminate in the creation of a exportable cognitive computing continuum. The architecture provides the all the software components to enable other use cases as it implements a cognitive plane, a universal execution layer, and AI-enabled infrastructure. The deliverable have also introduced the early PoC prototypes for the use cases, outlined the different functional requirements and KPIs, and finally provided early results of the different technologies.

## 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] "Intel software guard extensions," 2022.
- [3] 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.
- [4] 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.
- [5] N. Computing, "Nearbyone edge orchestrator." <https://www.nearbycomputing.com/wp-content/uploads/2021/08/NearbyOne-Product-Data-Sheet-v2.0.pdf>, 2021.
- [6] 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.
- [7] The Linux Foundation, "Kubernetes." <https://kubernetes.io/>, 2020.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] "Prometheus," 2023.
- [13] 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.
- [14] P. Liu, G. Bravo-Rocca, J. Guitart, A. Dholakia, D. Ellison, and M. Hodak, "Scanflow: An end-to-end agent-based autonomic ml workflow manager for clusters," in Proceedings of the 22nd International Middleware Conference: Demos and Posters, Middleware '21, (New York, NY, USA), p. 1–2, Association for Computing Machinery, 2021.
- [15] P. Liu, G. Bravo-Rocca, J. Guitart, A. Dholakia, D. Ellison, and M. Hodak, "Scanflow-k8s: Agent-based framework for autonomic management and supervision of ML workflows in kubernetes clusters," in 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 376–385, 2022.
- [16] "Pravega." <https://cncf.pravega.io>.

- [17] R. Gracia-Tinedo, F. Junqueira, T. Kaitchuck, and S. Joshi, "Pravega: A tiered storage system for data streams," in Proceedings of the 24th International Middleware Conference, pp. 165–177, 2023.
- [18] "Gstreamer." <https://www.nct-heidelberg.de/en/the-nct.html>, 2024.
- [19] "Pravega - gstreamer connector." <https://github.com/pravega/gstreamer-pravega>, 2024.
- [20] R. Gracia-Tinedo, F. Junqueira, B. Zhou, Y. Xiong, and L. Liu, "Practical storage-compute elasticity for stream data processing," in Proceedings of the 24th International Middleware Conference: Industrial Track, pp. 1–7, 2023.
- [21] Y. Yang, L. Zhao, Y. Li, H. Zhang, J. Li, M. Zhao, X. Chen, and K. Li, "Infless: a native serverless system for low-latency, high-throughput inference," in 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22), (New York, NY, USA), pp. 768–781, Association for Computing Machinery, 2022.
- [22] A. P. Twinanda, S. Shehata, D. Mutter, J. Marescaux, M. De Mathelin, and N. Padoy, "Endonet: a deep architecture for recognition tasks on laparoscopic videos," IEEE Transactions on Medical Imaging, vol. 36, no. 1, pp. 86–97, 2016.
- [23] E. Caron and R. Gracia-Tinedo, "The nanoservices framework: Co-locating microservices in the cloud-edge continuum," in 2023 IEEE 31st International Conference on Network Protocols (ICNP), pp. 1–6, IEEE, 2023.
- [24] "Greasy software user's guide." <https://github.com/rancher/local-path-provisioner>, 2023.