TRANSPARENT APPLICATION DEPLOYMENT IN A SECURE, ACCELERATED AND COGNITIVE CLOUD CONTINUUM

Grant Agreement no. 101017168

Deliverable D2.5
Final Version of SERRANO Architecture

Programme:	H2020-ICT-2020-2
Project number:	101017168
Project acronym:	SERRANO
Start/End date:	01/01/2021 – 31/12/2023

Deliverable type:	Report
Related WP:	WP2
Responsible Editor:	ICCS
Due date:	30/06/2022
Actual submission date:	30/06/2022

Dissemination level:	Public
Revision:	FINAL

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017168

Revision History

Date	Editor	Status	Version	Changes
02.05.22	ICSS	Draft	0.1	Initial ToC and content based on D2.3
31.05.22	ICCS	Draft	0.2	Updates in Sections 4.2, 4.4, 8.2
02.06.22	ICCS	Draft	0.3	Updates in Sections 2, 9
10.06.22	IDEKO	Draft	0.4	Updates in Section 7.3
10.06.22	CC	Draft	0.5	Updates in Sections 3.2.1, 7.1
14.06.22	AUTH	Draft	0.6	Updates in Sections 3.1.1, 4.3.1, 4.3.2, 5.2.2.2 and 6.3
16.06.22	UVT	Draft	0.7	Updates in Section 4.3.5
17.06.22	INNOV	Draft	0.8	Updates in Sections 4.1, 5.2.1 and 6.1
17.06.22	HLRS	Draft	0.9	Updates in Sections 5.2.2 and 5.2.2.1
20.06.22	ICCS	Draft	0.10	Consolidated version including updates in Sections 5 and 6
21.06.22	NBFC	Draft	0.11	Updates in Sections 5.2.2.2 and 6
21.06.22	ICCS	Draft	0.12	Integrate content by INTRA, HLRS, CC in Sections 5.2, 6 and 8.1
27.06.22	ICCS	Draft	0.13	Consolidated version addressing internal review comments.
30.06.22	ICCS	Final	1.0	

Author List

Organization	Author
ICCS	Aristotelis Kretsis, Panagiotis Kokkinos, Polyzois Soumplis, Emmanouel Varvarigos
MLNX	Juan Jose Vegas Olmos
CC	Marton Sipos, Daniel Lucani
USTUTT/HLRS	Javad Fadaie Ghotbi and Kamil Tokmakov.
AUTH	Kostas Siozios, Dimosthenis Masouros, Dimitrios Danopoulos, Ioannis Oroutzoglou, Argyris Kokkinis, Aggelos Ferikoglou
INTRA	Paraskevas Bourgos, Makis Karadimas
INB	Ferad Zyulkyarov
INNOV	Andreas Litke, Stelios Pantelopoulos, Filia Filippou
IDEKO	Aitor Fernández, Javier Martín
UVT	Gabriel Iuhasz, Silviu Panica, Florin Adrian Spataru
NBFC	Anastasios Nanos, Charalampos Mainas, George Ntoutsos, Kostis Papazafeiropoulos, Ilias Apalodimas

Internal Reviewers

Ioannis Oroutzoglou, Aggelos Ferikoglou, AUTH
Juan Jose Vegas Olmos, MLNX

Abstract: This deliverable (D2.5) concludes the outcomes of *Task 2.3 "Architecture Specification"*, *Work Package 2 "Requirements and System Design"* of the SERRANO project. The deliverable presents the final architecture specifications of the SERRANO platform based on the feedback from the development and integration activities within the other technical Work Packages (3-6). It also presents the platform's key building blocks and describes their interactions along with the required interfaces. Moreover, it includes the updated version of the SERRANO implementation and delivery plan for the second iteration (M19-M36) of the implementation towards the final release of the SERRANO platform (M36).

Keywords: SERRANO architecture, SERRANO platform, transparent deployment, hardware acceleration, secure storage, cognitive orchestration, service assurance

Table of Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
ASGI	Asynchronous Server Gateway Interface
BDI	Belief Desire Intention
BRAM	Block Random Access Memories
CDC	Change Data Capture
CFD	Computational Fluid Dynamics
CI/CD	Continuous integration/Continuous Deployment
CNC	Computer Numerical Control
CORS	Cross-Origin Resource Sharing
CSV	Comma-Separated Values
D	Deliverable
DCGM	Data Center GPU Manager
DOCA	Data center On a Chip Architecture
DPU	Data Processing Unit
DSP	Digital Signal Processing
EaaS	Everything as a Service
EDE	Event Detection Engine
EU	European Union
FaaS	Function as a Service
FCE	FPGA Exporter
FF	Flip Flops
FPGA	Field Programmable Gate Array
GA	Genetic Algorithm
GCP	Google Cloud Platform
GPU	Graphics Processing Unit
HLRS	High Performance Computing Center Stuttgart
HLS	High-Level Synthesis
HPC	High Performance Computing
HW	Hardware
IO	Input/Output
K3s	Lightweight Kubernetes
K8s	Kubernetes
LUT	Lookup Table
M	Month
MAAS	Metal as a Service
MC	Monte-Carlo
ML	Machine Learning
MOM	Message-Oriented Middleware
MPI	Message Passing Interface
NUMA	Non-Uniform Memory Access
OCI	Open Container Initiative
OSS	Object Storage Server
OST	Object Storage Target

PCIe	Peripheral Component Interconnect Express
PXE	Preboot Execution Environment
QoS	Quality-of-Service
RDMA	Remote Direct Memory Access
REST	Representational State Transfer
RLNC	Random Linear Network Coding
RoCE	RDMA over Converged Ethernet
ROT	Resource Optimization Toolkit
RPC	Remote Procedure Call
S2S	Source-to-Source
SAR	Service Assurance and Remediation
SDK	Software Development Kit
SEV	Secure Encrypted Virtualization
SFTP	Secure File Transfer Protocol
SGX	Software Guard Extensions
SLA	Service Level Agreement
SW	Software
TLS	Transport Layer Security
TPM	Trusted Platform Module
UC	Use Case
URL	Uniform Resource Locator
VM	Virtual Machine
VMM	virtual Machine Monitor
VVUQ	Verification, Validation and Uncertainty Quantification
WP	Work Package
WSGI	Web Server Gateway Interface

1 Executive Summary

SERRANO envisages the development and deployment of disaggregated federated cloud infrastructures that operate, process and store on edge, enabling accelerated edge nodes as integral parts of the computation and storage chain. In addition, the SERRANO ecosystem expansion includes HPC infrastructures that can be utilized for exceptionally computationally intensive simulations and data analysis, bridging the gap between these currently largely separated computing paradigms.

Deliverable D2.5 “Final version of SERRANO Architecture” provides the final reference architecture of the SERRANO platform as was refined based on the feedback from the development activities during the first iteration of the implementation and the analysis of the initial version of the SERRANO platform (M1-M18). Moreover, the innovations and technological advancements of the SERRANO project are highlighted, providing information on the novel ecosystem of cloud-based technologies, spanning from specialized hardware resources up to software toolsets that the SERRANO utilizes and develops. A more detailed description of SERRANO-enabled technologies is available at the respective technical deliverables from Work Packages 3-5.

The multi-layer architecture of the SERRANO platform integrates individual functionalities and features that are being developed to create a single borderless infrastructure. The selected modular architecture enables the implementation of future extensions and the individual use and exploitation of platform components. Furthermore, the document presents a set of workflows that highlight the interactions and exchange of information between the core components and services of the SERRANO platform. The deliverable also specifies the required vertical and horizontal interfaces that enable the communication between the main components.

The information provided in the present deliverable is expected to comprise a guideline framework for the rest of the project’s Work Packages during the second iteration of the implementation plan (M19-M36) towards the provision of the final release of the SERRANO platform.

2 Introduction

SERRANO targets the efficient and transparent integration of heterogeneous resources, providing an infrastructure that goes beyond the scope of the "typical" cloud and realizes a true computing continuum. The project aims to define an intent-based paradigm of operating federated infrastructures consisting of edge, cloud, and HPC resources, which will be realized through the SERRANO platform (Figure 1). SERRANO will automate the process of application deployment across the various computing technologies, translating applications' high-level requirements to infrastructure-aware configuration parameters. Next, the SERRANO platform will determine the most appropriate resources of the cloud continuum to be used by an application, and then transparently deploy workloads and coordinate data movement. Also, SERRANO will continuously adapt the deployed applications, based on the observe, decide, act approach.

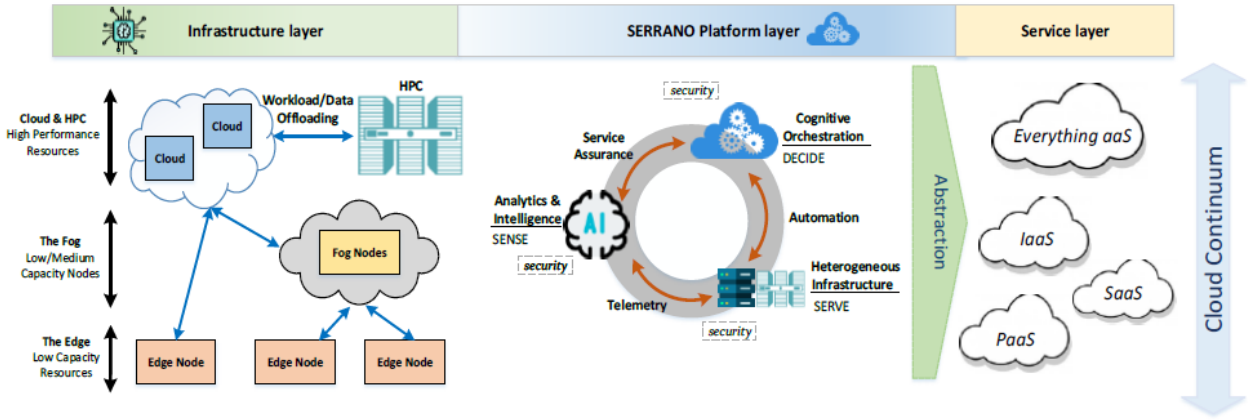


Figure 1: The SERRANO platform, utilizing edge, cloud and HPC resources and empowering the Everything as a Service (EaaS) notion towards the cloud continuum

2.1 Purpose of this document

The present deliverable (D2.5) presents the final outcomes of Task 2.3 – "Concept, Use Case Requirements and Business Prospects" of Work Package 2 – "Requirements and System Design of the SERRANO project". T2.3 is associated with the design of the overall SERRANO architecture and the definition of the vertical and horizontal interfaces between the SERRANO components.

D2.5 builds on the initial architecture specification and final contributions of Task 2.2, as reported in the respective deliverables (D2.3 in M9 and D2.4 in M16), towards the final version of SERRANO architecture. Also, the deliverable presents the final high-level architecture of the SERRANO platform, the interactions among the platform's key building blocks, and the associated interfaces. This deliverable also presents an updated description of the SERRANO platform deployments to support the project's use cases (UCs). A more comprehensive description of these activities is available in deliverables D6.1 "Use cases technological developments" (M16) and D6.2 "KPIs and evaluation methodology" (M18).

In the next period, SERRANO enters the second iteration of the implementation plan. To this end, D2.5 will serve as a guideline for the remaining research and development activity conducted in the technical work packages (WP3-5) and the UCs development, evaluation, and integration activities under WP6. The technical work packages will have to ensure that project's developments are fully aligned with the specifications of the final architecture and interfaces.

2.2 Document structure

The present deliverable is split into ten major chapters:

- Executive Summary
- Introduction
- SERRANO Technologies and Resources
- SERRANO Platform Components
- SERRANO Architecture
- Interfaces Specification
- SERRANO Platform Deployment View
- SERRANO Implementation and Delivery Plan
- Requirements Coverage
- Conclusions

2.3 Audience

This document is publicly available and should be helpful to anyone interested in the final specification of the overall SERRANO architecture, the respective interfaces, and workflows. Moreover, this document can also be useful to the general public for obtaining a better understanding of the framework and scope of the SERRANO project.

3 SERRANO Technologies and Resources

SERRANO introduces a novel ecosystem of cloud-based hardware and software technologies corresponding to SERRANO-enhanced infrastructure resources (Figure 2). The SERRANO platform integrates novel hardware and software technologies and methodologies towards an application-optimized and secure service instantiation. These mechanisms include: (i) hardware acceleration for encrypted storage, (ii) multi-level approximate hardware accelerators for dynamic and input-driven approximations, (iii) scalable distributed secure storage, (iv) trusted execution and workload isolation, (v) lightweight virtualization for seamless deployment in cloud and edge, (vi) software kernels for computationally intensive tasks acceleration and (vii) hardware acceleration abstractions.

Figure 2: SERRANO-enhanced hardware and software resources

In the following sections, we provide details for the SERRANO's hardware and software resources and associated technologies.

3.1 SERRANO Hardware Resources

3.1.1 Hardware Accelerators: FPGA and GPU

SERRANO's resources for hardware acceleration at the edge and cloud remain the same as those described in deliverable D2.3 (Section 3.1.1). Besides that, two more Cloud GPU cards have been donated from NVIDIA/MLNX and integrated on AUTH's premises. In short, in the following table, we describe the final list of acceleration devices.

Table 1: Hardware cloud and edge acceleration devices

Edge	Xilinx ZCU102 FPGA device
	Xilinx ZCU104 FPGA device
	NVIDIA Xavier NX GPU device
	NVIDIA Xavier AGX GPU device
Cloud	Xilinx Alveo U50 FPGA device
	Xilinx Alveo U200 FPGA device
	2x NVIDIA T4 GPU devices

3.1.2 Smart NICS and Data Processing Units

NVIDIA Mellanox ConnectX® SmartNICs utilize stateless offload engines, overlay networks, and native hardware support for RoCE and GPUDirect™ technologies to maximize application performance and data center efficiency. Developers can use ConnectX custom packet processing technologies to accelerate server-based networking functions and offload datapath processing for compute-intensive workloads, including network virtualization, security, and storage functionalities.

The NVIDIA® BlueField®-2 data processing unit (DPU) is the world's first data center infrastructure-on-a-chip optimized for traditional enterprises' modern cloud workloads and high-performance computing. It delivers a broad set of accelerated software defined networking, storage, security, and management services with the ability to offload, accelerate and isolate data center infrastructure. With its 200Gb/s Ethernet or InfiniBand connectivity, the BlueField-2 DPU enables organizations to transform their IT infrastructures into state-of-the-art data centers that are accelerated, fully programmable, and armed with "zero trust" security to prevent data breaches and cyber-attacks. By combining the industry-leading NVIDIA ConnectX®-6 Dx network adapter with an array of Arm® cores and infrastructure-specific offloads, BlueField-2 offers purpose built, hardware-acceleration engines with full software programmability. Sitting at the edge of every server, BlueField-2 empowers agile, secured and high-performance cloud and artificial intelligence (AI) workloads, all while reducing the total cost of ownership and increasing data center efficiency. The NVIDIA DOCA™ software framework enables developers to rapidly create applications and services for the BlueField-2 DPU. NVIDIA DOCA makes it easy to leverage DPU hardware accelerators, providing breakthrough data center performance, efficiency and security.

Figure 3: BlueField2 DPU: 8 ARM A72 CPUs, 8MB L2 cache, 6MB L3 cache in 4 Tiles, ARM Frequency: 2.0-2.5GHz, Dual 10-100Gb/s Ethernet & InfiniBand single 200Gb/s, PCIe gen4.

3.1.3 High Performance Computing

The High Performance Computing Center Stuttgart (HLRS) provides the computational resources that allow scientists and engineers to solve complex problems. It operates various HPC systems (see [1]) for a wide range of HPC applications. The HPE Apollo 9000 Hawk system is the most powerful of them. At the time of installation, it was among the fastest high-performance computers in the world and the fastest general-purpose machine for industrial production in Europe. In Top500's November 2020 list, the Hawk ranked 16th for HPL and 18th for HPCG [2].

The computing power of the supercomputers results from the higher number of compute nodes, high-performance interconnect and performance-optimized software. The high cost of the HPC infrastructure requires efficient use of the supercomputers. This is an important condition for integrating the HPC services in the SERRANO project and imposes certain requirements on the use cases of the project and the SERRANO Resource Orchestrator.

Figure 4 on the right side shows a schematic representation of a 4-dimensional segment of a 9-dimensional hypercube topology [3]. A node in the graph represents an InfiniBand-Switch. One link supports the data transfer with theoretically 200 Gbits/s. Each InfiniBand-Switch connects 16 compute nodes. Therefore, a 3D cube represents one of 44 racks. As the reader can see, the rack switches are connected along three dimensions. The switches of the different racks are interconnected along the additional dimensions of a hypercube. Only 4 of 9 dimensions are shown. The links across the other dimensions connect a total of 44 racks with 5632 nodes. The ninth dimension is not fully implemented. For the complete implementation of a 9D-hyper cube the number of nodes must be 8192. The implementation of Hawk's interconnect required 3,024 cables with a total length of 20 km.

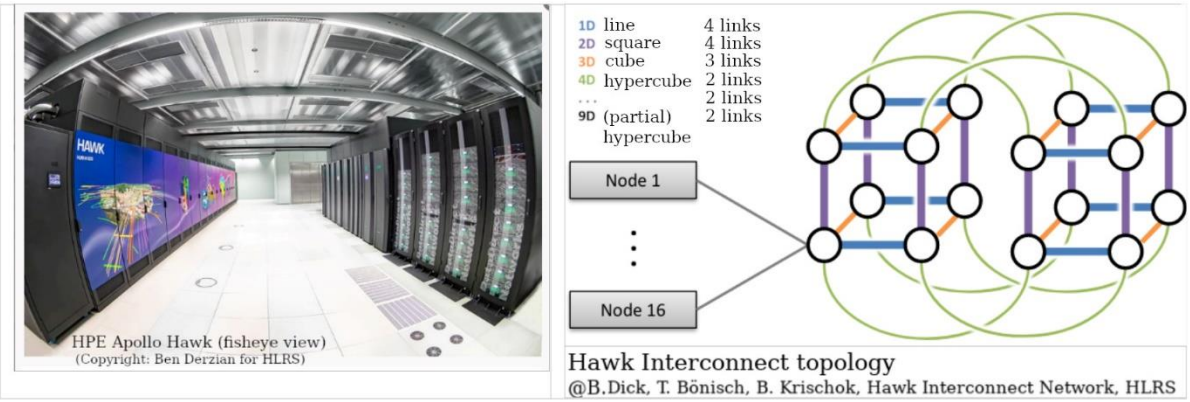


Figure 4: On the left, 20 of totally 44 cabinets of the supercomputer Hawk.. On the right, part of 9D-Hypercube internetwork topology graph (4D hypercube).

Due to the excessive cost of the interconnect and despite large aggregated bandwidth but hierarchically limited link density between processing units, both the HPC Resource Manager (PBS system) and the applications must consider the underlying network topology to avoid the bottlenecks. For example, network congestion at one or more switches or increased latency between communicating internodes due to multiple hops across the different dimensions of the network topology. Table 2 describes the main components of the Hawk.

Table 2: Main parameters of HPE Apollo (Hawk) HPC System @ HLRS

Number of cabinets	44	**CPUs per node**	2
Number of compute nodes	5,632	**Cores per CPU**	64
System peak performance	26 Petaflops	**Number of compute cores**	720,896
Interconnect topology	Enh.9D-Hypercube	**CPU frequency**	2.25 GHz
Workspace (lustre)	~25 PB	**DIMMs in system**	90,112
Power consump. (Avrg./HPL)	~3.5 MW/ ~4.1 MW	**Total system memory**	~ 1.44 PB

Compute nodes

The Hawk's dual-socket computing nodes are equipped with AMD's "Rome EPYC 7742" processor [2] and 256 GiB RAM (DDR4-SDRAM@3200MHz). A compute node thus has 128 cores, which can be boosted to up to 3.4 GHz. Together with eight (8) memory channels, this provides a powerful computing unit. On the other hand, the newer CPU architectures are more sophisticated. For this reason, HPC applications require fine-tuning to make efficient use of the underlying hardware.

This is also confirmed by the results of a benchmark in Figure 5 on dual-socket compute nodes equipped with different processors, from “Intel Sandy Bridge” and “Intel Haswell” to “Intel Skylake” and “AMD EPYC Rome”.

The benchmark calculates the second derivative of a twice differentiable function “f(x,y,z)” on a three-dimensional domain of size 1024x1024x512. The discretization scheme of the algorithm uses a regular grid and a 27-point stencil for 3D Laplace operator [4]. Two parallel implementations with OpenMP-threads were tested. The difference between the implementations is that the three-dimensional array of size 1024x1024x512 with the discrete function values are arranged in blocks, e.g., of the size 64x64x128. This increases the locality of the data accesses, which significantly increases the efficiency of cache reuse and thus the performance of the algorithm. The results of the simple implementation are shown with the dashed curves. The blocked version is shown with solid lines. The diagram clearly demonstrates that the relative differences in the performance of the two algorithm versions (Simple3d and Block3d) for the older "Intel Sandy-bridge" or "Intel Haswell" processors are significantly lower than the relative differences in the performance of the two versions for the newer “Intel Skylake” or “AMD EPYC Rome” processors.

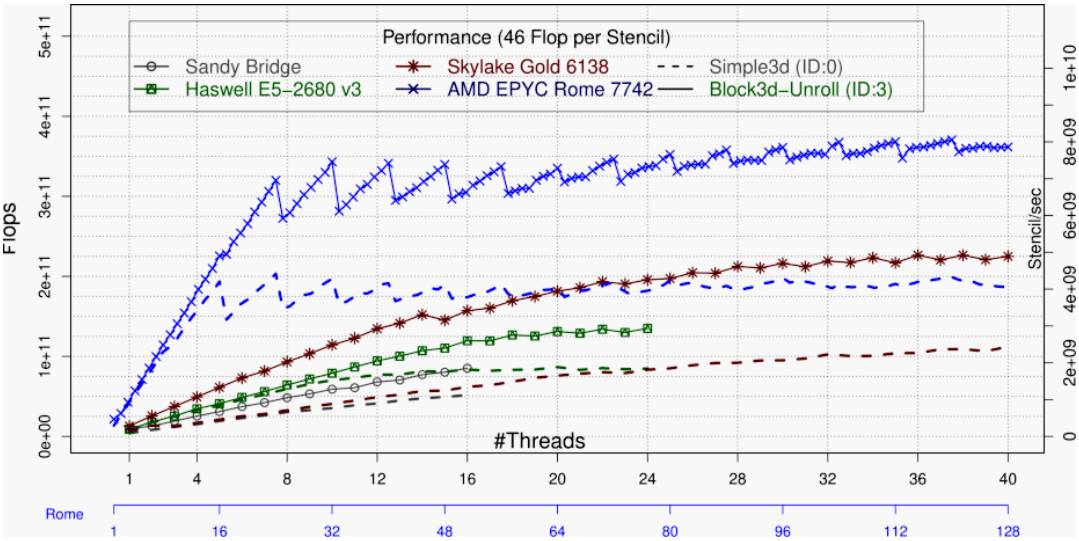


Figure 5: Performance of the different implementations of the 3D Laplace- operator on dual systems

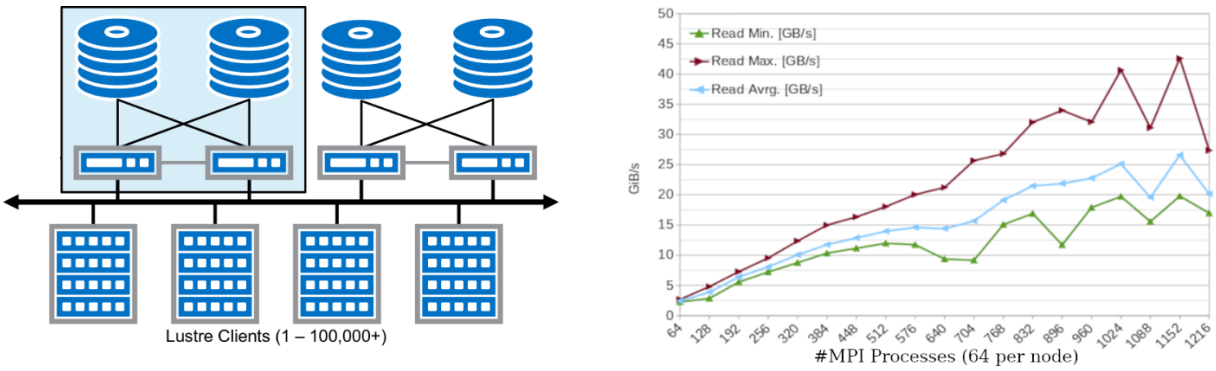
Since the number of nodes is limited and power consumption of the HPC system is high, it is important to perform a hardware specific optimization of the applications.

IO-Workspace

As described in D2.2 "SERRANO use cases, platform requirements and KPIs analysis", the HPC file system for parallel input/output (IO) operations is a shared and limited resource. Figure 6 shows a schematic of a Lustre file system (www.lustre.org). The file system architecture allows parallel access to the multiple raids of the hard disks to read or write the files by multiple clients in parallel. The raids are grouped together by Object Storage Targets (OSTs), which are worked independently from each another and connected to high performance network via Object Storage Servers. Hence, all OSTs can be accessed from every computing node of the supercomputer as parallel as possible.

The results of an IO benchmark are shown on the right side of the figure. A weak scaling experiment was performed for the parallel IO operation "MPI_File_read_all". Each process reads 16 MiB of data. The data is distributed across 16 OSTs with a chunk size of 1 MiB. The measurements show the minimum maximum and average bandwidth during the execution of the IO-Read operation on 64 (one compute node) up to 1216 MPI (19 compute nodes) processes. Every second core of the involved nodes was used. On the one hand, the bandwidth scales with the increasing number of processors and consequently the amount of data to be read. On the other hand, the results show a high variation. One of the main reasons for this is that in contrast to a compute node and its memory, the IO resources are limited and shared and used by multiple applications simultaneously.

Hence, HPC systems are designed for tasks that require significant computing power, a lot of main memory, appropriate amount of communication and less IO. More details about the Hawk architecture can be found in [1].



(a) (b)

Figure 6: (a) The basic schematic of the Lustre parallel high-performance file system (www.lustre.org), (b) The results of the IO bandwidth benchmark for the Hawk parallel file system (type Lustre).

3.1.3.1 Near Future Extensions of HPC Resources at HLRS

The HPC environment of HLRS are continuously being enhanced to increase stability and support different user's workflows. For example, Hawk is being currently extended with several GPU compute racks, which will be fully integrated with Hawk via HDR Fabric and therefore have access to both "Quobyte" object system and Hawk's workspace. This enables for example support for HPC/AI workflows.

Figure 7 shows two examples of the workflows, which includes both an HPC and AI parts:

A. HPC simulations executed on Hawk's compute nodes can produce training sets for the AI part of the workflow and stores it in the workspace. This can also be used for the automatic detection of certain features, such as intense turbulences, in the results of Computational fluid dynamics (CFD) simulations.

B. AI algorithm generates the sets of the configuration parameters for the series of CFD simulations, for example, to find the optimal shape of a simulated wing.

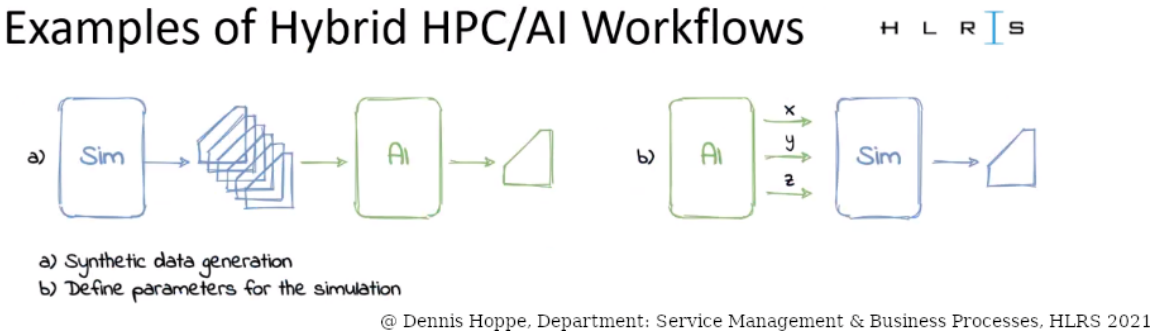


Figure 7: Examples of Hybrid HPC/AI Workflows.

3.1.4 General edge hardware

3.1.4.1 SmartBox

IDEKO works closely with Savvy Data Systems, a technological start-up focused on machine-monitoring and data analytics. In conjunction with them, IDEKO has developed the Smart Box, an industry-ready box for gathering machine data. The Smart Box is a data gathering and data gateway (Debian Jessie host, 4 - 8GB RAM, 60GB HD, Celeron Quadcore 1.6GHz - 2.09GHz.), an industry PC for gathering machine data that can connect to the most common CNC models (machines) and other data origins and sensors.

There are different models of boxes with different price ranges and computational power. Boxes are attached to machines with 1 to 1 relation. This leads to a scenario where some machines have more computational power than others just because the box attached is more powerful.

The Smart Box already has all the connectivity needed to get data out of the machine and send it to a private cloud. The box can read machine indicators (axis positions, oil pressure, the running program, etc.) at a configured frequency, usually every second. Moreover, it is almost plug-and-play and supports remote configuration as well as remote upgrades

The box allows the deployment of Docker containers. We use the containers for developing edge-computing solutions or when the customer does not allow us to store their data in the cloud.

3.1.4.2 On-premises storage gateway and edge resources

The On-premises storage gateway will be the key component of the SERRANO-enhanced storage service developed by Chocolate Cloud. It is designed to be deployed on a company's existing infrastructure. To simplify this process and make it as versatile as possible, it will be a containerized application, running on existing available commodity hardware.

On the other hand, it will be tailored to take full advantage of SERRANO-enhanced hardware, when available. It will use hardware acceleration for encrypting TLS connections by leveraging MLNX Bluefield SmartNICs, described in Section 3.1.2. This is done to reduce load on the CPU and thus increase scalability in terms of the number of supported concurrent connections. The erasure coding, encryption, and potentially compression of data will be accelerated using FPGAs and GPUs, described in Section 3.1.1. Beyond offloading these computations from the CPU, we also expect a reduction in the service's response time. The Gateway's hardware requirements depend on the number of concurrent requests that it must support. At a minimum, it should be deployed on a desktop-class computer with 2 cores and 4 GB of RAM.

The second component of the storage service will be a set of on-premises storage locations. We refer to these as SERRANO edge devices and will be in essence a set of object stores with

either S3 or SWIFT-compatible API. Similarly, to the Gateway, these will also be deployed as containerized applications. The SERRANO edge devices will rely on the local file system to store data. As such, we will most likely use the Persistent Volumes feature of Docker. This component's hardware requirements are more likely to be more modest, ideally requiring at least a single core and 2 GB of RAM.

3.1.4.3 FinTech processing

InbestME (INB) and the FinTech use case does not utilize any special edge hardware. INB infrastructure consists of general-purpose computers, some of which are equipped with GPUs. Some of the computers are hosted on-premises and others on the cloud. The INB software system consists of a combination of web applications, API services, microservices, containers, and standalone applications. INB will benefit from the SERRANO service deployment (Section 3.2.4) and acceleration (Section 3.2.3) by migrating its solution to microservices and containers as well as abstracting function as a service (FaaS). INB will explore a deployment in which reusable functions such as portfolio analysis will be made available through FaaS. The FaaS will be implemented through microservices that run inside containers or lightweight unikernels. It will be possible to scale the system by instantiating more containers depending on the load. To benefit from the acceleration and the available resources the SERRANO platform will schedule the containers in smart and transparent manner.

3.1.4.4 Virtualization HW dependencies

Virtualization is a key component of facilitating the deployment of workloads on the SERRANO platform. To alleviate the overhead of emulation, hardware extensions for virtualization support are crucial. To this end, the chosen hardware components provide the necessary hardware mechanisms to spawn Virtual Machines: Virtualization support for ARM-based nodes is available since the ARMv8 spec; for the x86 equivalent platforms AMD/Intel support is present on all modern processors.

3.1.4.5 EDGE SandBox

UVT will focus on optimizing the event detection toolkit to efficiently run on less powerful devices that are normally used on the edge. UVT is setting up a mini-EDGE devices cluster (custom made, presented in Figure 9) using NVIDIA Jetson Nano [5] development boards for testing and development purposes. These are small but powerful computers that allows you to run multiple neural networks in parallel for various applications that involve Machine Learning techniques or algorithms.

The mini cluster at UVT consists of 10 devices with the following specifications:

- **CPU:** Quad-Core ARM A57 @ 1.43 GHz;
- **GPU:** 128-core Maxwell;
- **Memory:** 4GB 64-bit LPDDR4;
- **Storage:** microSD and 256GB SATA3 USB3.0 SSD;

- **Connectivity:** 1Gbps Ethernet;
- **USB:** 4x USB3.0, 1x USB 2.0 Micro-B;
- **Other capabilities:** video encode: 4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265), video decode: 4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265).

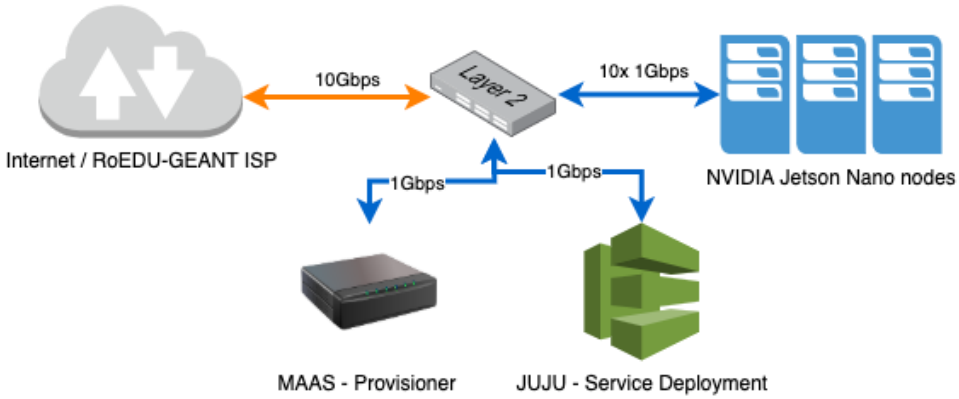


Figure 8: UVT Jetson Nano Cluster Setup

Each Jetson Nano device is set up to boot from the attached SSD drive and has a 1Gbps network connection for external access. The SSD drive has three independent partitions: (1) boot required kernels and ram disks, (2) root file system and (3) recovery partition. The device initially tries to boot from the network using PXE boot and as a failover will boot the local kernel and ramdisk. The recovery partition is used to restore to factory defaults a broken operating system by selecting specific boot parameters, either remotely (via PXE boot) or locally (by rebooting the operating system). In Figure 8, the cluster setup is depicted. The Jetson Nano devices are administered by a MAAS [6] bare metal provisioning service that controls and bootstraps the basic operating system on the device's hard drive. Further, the JUJU [7] deployment service will customize the operating system by deploying the desired software stack. In the context of SERRANO, we will use the lightweight version of Kubernetes, K3s [8], as an orchestrator for different software components that we plan to develop and test in the edge infrastructure. Of course, leveraging the bespoken technologies, the cluster setup can be easily adapted for other development scenarios. Also, the cluster can be split-up to create multiple individual mini-clusters to accommodate various testing and development edge-like environments.

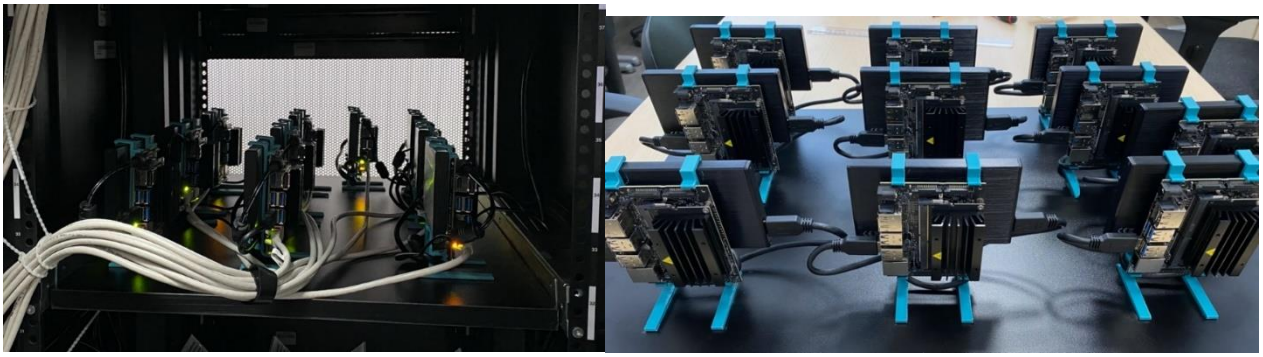


Figure 9: UVT Jeston Nano Cluster - Rack Mount

3.2 SERRANO Software Resources

3.2.1 SERRANO-enhanced Storage Service

Chocolate Cloud will develop the SERRANO-enhanced Storage Service as part of Work Package 3, motivated by the requirements of Use Case 1: Secure Storage. Deliverable 2.4 includes a detailed presentation of the service's key features, which user requirements it seeks to meet as well as an overview of the components and their relationships. In this document, we focus on the technical specification of each component.

The SERRANO-enhanced Storage Service is built around SkyFlok, a secure file storage and sharing solution. SkyFlok lets users select the cloud providers and where exactly on the globe their data should be stored. Files are protected using encryption and a novel erasure code called Random Linear Network Coding (RLNC) both from malicious third parties and curious cloud providers. Thanks to its multi-cloud approach, the system can maintain data availability and reliability even if one or more cloud locations experience an outage or permanent data loss.

The SERRANO-enhanced Storage Service extends SkyFlok, taking it from a cloud-only solution to one stretching to the edge. To this end, it offers edge storage locations on SERRANO edge devices and an on-premises storage gateway tailored explicitly to enterprise users. In addition, as a part of the SERRANO platform, the service takes advantage of platform features and services such as orchestration to deliver automated storage policy creation/assignment (described in D2.4) and hardware acceleration possibilities as described in Section 3.1.4.2. A high-level view of the service's components is presented in Figure 10.

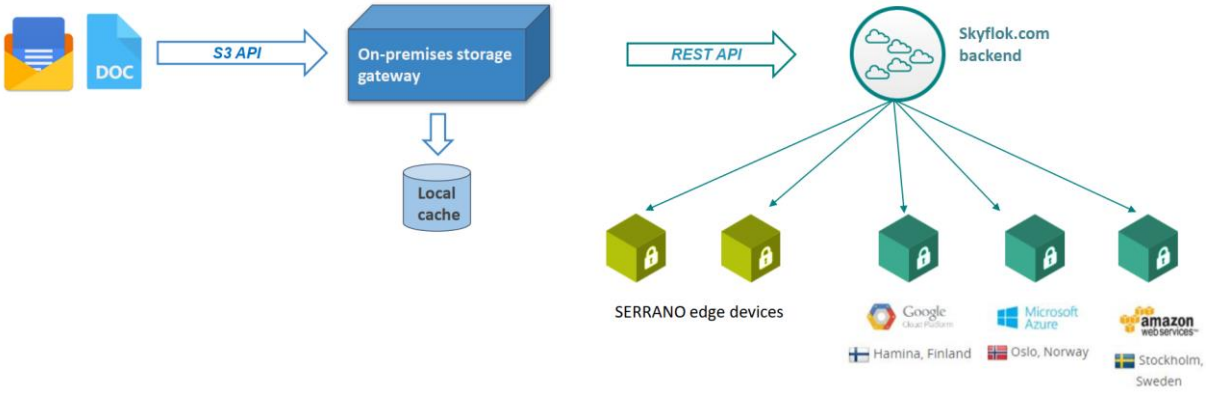


Figure 10: The components of the SERRANO-enhanced Storage Service

The Skyflok.com backend has been developed before the SERRANO project started. It is composed of a set of micro-services, hosted on Google App Engine. During the project, modifications to the backend will be kept to a minimum as the SERRANO-specific features will be provided through the two services described below. However, some changes will inevitably be needed, including a new API designed to provide information about the state and general characteristics of the cloud and edge storage locations.

3.2.1.1 On-premises storage gateway

The on-premises storage gateway (Gateway from now on) will be the principal component of the service. It will provide an S3-compatible API to the applications and services of the SERRANO platform.

The Gateway will be developed from scratch for SERRANO using Python 3.8. It will be created using FastAPI, a web framework built to reliably serve a large number of HTTP requests simultaneously. Flask is a more established solution with good support and a large community of developers that use and extend it. However, FastAPI is a much newer framework and as such uses many of Python's newest features and libraries. This should give it an edge in both performance [9] and ease of development. While both support asynchronous operations, FastAPI has been built from the ground up to use coroutines when serving requests. Whereas Flask uses the well-known WSGI (Web Server Gateway Interface) interface to interact with a web server such as nginx, FastAPI relies on its spiritual successor, ASGI (Asynchronous Server Gateway Interface) [10]. The downside of the Flask approach is that every request ties up a worker thread, even if the endpoint is served by an async function. Given that the Gateway will perform a large number of I/O operations, with HTTP requests to cloud locations taking a considerable time, efficient asynchronous I/O is crucial to achieving a good user experience. Chocolate Cloud has assessed both options through deep-dive prototypes and selected FastAPI.

The Gateway will feature data processing in three notable ways: encryption, erasure coding, and compression. There are several mature Python libraries that offer both complete cipher implementations and cryptographic primitives. The same is not true for erasure coding, especially if we consider that SkyFlok uses Random Linear Network Coding, a relatively new and less know technique. As such, we will either use a CPython-based wrapper around Kodo [11], a high-performance C++ implementation or develop our own Python-native implementation. The decision will be based on several factors, including performance, the potential for acceleration, licensing and so on. Compression will be implemented using Python libraries available through PIP.

To enable simple and flexible deployment, the Gateway will be a containerized application. It will maintain a minimal state, with a key exception being the caching layer. To reduce the response time of read and write operations, the Gateway will feature a caching function that will take advantage of the fact that it will be deployed on the customer's premises. This allows for a simple Least Recently Used write-through cache to be efficient.

3.2.1.2 SERRANO edge devices

The second component designed to enable SkyFlok to be extended to the edge will be the SERRANO edge devices. These devices will provide low-latency on-premises storage locations where SERRANO users can distribute their files. At a high level, this component should be fairly simple. It must provide a simple storage API through which the Gateway can store and retrieve erasure coded fragments of user data. The component must be simple to deploy and monitor.

It does not need to provide reliability through redundancy on its own, since this is the task of the storage service.

SkyFlok currently stores coded fragments in cloud-based object stores. As such, the Skyflok.com backend has connectors to both the proprietary interfaces (e.g., Google Cloud Platform, Azure) and the more standard interfaces (S3) as well as to the commonly deployed OpenStack Swift's API. While the SERRANO edge devices do not need to have the complete feature set of object storage solutions, it makes sense to provide part of an object-store-like storage API to facilitate easy integration with the Skyflok.com backend. Given its widespread adoption, S3 is a good option. Details on this are presented in Section 6.2.

SkyFlok makes use of the concept of pre-signed URLs (temporary URLs in OpenStack Swift parlance) to connect its users directly with the data stored in the clouds. Whenever a file is to be uploaded or downloaded, the user's browser authenticates with the Skyflok.com backend and requests a set of upload/download links. The links point to protected resources on the cloud locations. This poses a security challenge; how can the user's browser be trusted with the credentials necessary to access these resources. The solution is to create a special set of cryptographically signed links – pre-signed URLs – which do not require additional credentials. The links have a short lifespan and can only be used once. The signature is calculated using asymmetric-key cryptography and is guaranteed to be different every time. Given the elegance of this solution, the SERRANO edge devices must support this feature. Figure 11 illustrates how a file is downloaded using a pre-signed URL. Furthermore, support for Cross-Origin Resource Sharing (CORS) is also needed to maintain browser-based access.

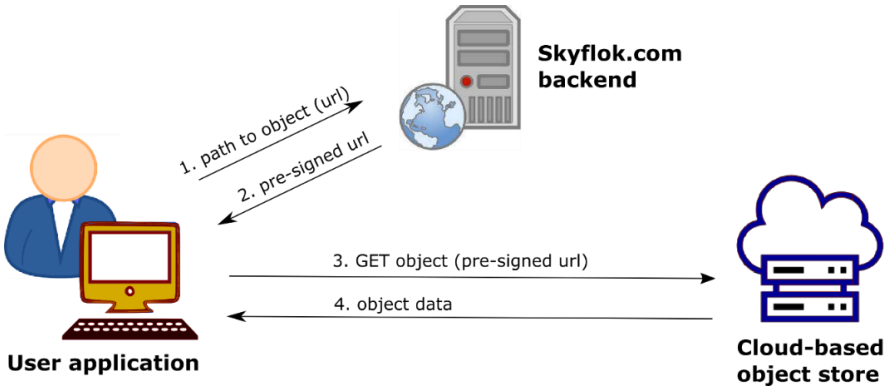


Figure 11: Using a pre-signed URL to download a file

Given the requirements above and those presented in D2.4, the choice is between using an off-the-shelf object store and a custom implementation. We assessed three different object stores: Ceph, OpenStack Swift and MinIO. Of the three, we favour MinIO, a lightweight solution, as its name suggests. It has fewer unnecessary features than the more established solutions and can also run in a non-distributed manner, with no redundant storage. It is being actively developed in Go and is open-source. Thus, we can, at least to a certain degree, tailor it to our requirements. It has a free Community edition based on the GNU AGPL v3 license.

We have ascertained, using a basic deployment and a set of tests, that it provides all S3 endpoints required by the Skyflok.com backend, including support for pre-signed URLs and

CORS. It has been designed to be run in a container, with all object data and metadata being stored on a virtual Docker volume. When running in a Kubernetes cluster, we can use its persistent volumes feature to deploy it inside an organization on any pod where this volume is mountable from.

Regarding security, it is simple to set up access key pairs used by the Skyflok.com backend to authenticate itself with MinIO deployments. This configuration can be done both from a web application's GUI as well as programmatically through the MinIO's REST API and directly in a Dockerfile when running inside a container. In addition, it features encryption at rest, configurable to be global or request-based. As a bonus, MinIO also runs on ARM systems, paving the way for it to be deployed on low-powered devices like Raspberry Pis. Based on our preliminary tests, MinIO appears as an excellent choice. However, we leave the option open for the future to create a custom Python service to serve SERRANO edge devices. We will choose this option in the unlikely scenario in which we cannot provide with MinIO the features necessary to integrate the Storage Service with the other SERRANO platform services.

3.2.2 Trust execution and workload isolation

To be able to move to a multi-tenancy execution model at the edge, the systems software must ensure non-interference and controlled data access among different and concurrently running applications. To this end, virtualization plays a significant role in adding secure multi-tenancy execution at the edge. Adding another layer of abstraction facilitates unified execution frameworks, but also complicates the execution stack and consumes resources to ensure correct management, isolation, and resource sharing among tenant workloads.

To alleviate the significant overhead of adding a full virtualization stack (hypervisor & VMs) at the edge, the research community has proposed the use of container technology. Nonetheless, this execution model increases the attack surface, as it exposes the full OS and runtime layer to any malicious or compromised application running in a container. Related works and critical security advisories have pointed out that containers are far too insecure for multi-tenancy. Recent works have introduced a new type of resource virtualization, bringing the benefits of isolation and secure execution without the burden of a full virtualization stack to support generic Operating Systems. To address the security issues that arise from multi-tenancy and remote environments, while efficiently utilizing resources, we tackle these two major issues separately. D3.3 provides a more comprehensive description of the initial developments.

Attack surface: SERRANO minimizes the attack surface for applications running at the edge by leveraging the minimal exposure of trusted code to the application by packaging applications in unikernels. A unikernel is a tiny piece of binary, able to run as a conventional operating system bundled with the application, containing only the bits and pieces needed for execution. For example, a web server packed as a unikernel contains the binary code for the actual web server, the OS layers to provide network and storage capabilities and the layers needed for bootstrapping the application and the interaction with the hypervisor. Specifically, we build on the solo5 [12] architecture to minimize the kernel code being accessed from user-space

applications and VMs images/unikernels. To this end, we develop an efficient and secure mechanism for applications to be executed as part of a self-contained machine image, keeping only the needed dependencies and reducing the execution and exposure of trusted/kernel code to the minimum

Trusted execution: We harden the hypervisor and Virtual Machine Monitor (VMM) framework using hardware and software techniques. Specifically, we employ a strict security attestation mechanism at the lowest level of the software stack (VMM and OS-glue to the application), to ensure that the workloads running are legitimate. The mechanisms used to initiate the root of trust and the secure storage of the encryption keys are provided from the hardware: TrustZone for ARM and Software Guard Extensions (SGX)/Secure Encrypted Virtualization (SEV) for Intel/AMD processors. Additionally, we enhance this framework with the trusted boot extensions the platform is offering (TPM or similar).

3.2.3 Hardware acceleration abstractions

The common way of using hardware accelerators in a distributed, cloud-managed environment is by assigning the hardware accelerator directly to the instance where the workload that needs acceleration is running. This method, however, entails cumbersome setups and complicated requirements for the management layer raising at the same time security concerns. In 5G-COMPLETE[1], NBFC introduced a lightweight API-remoting framework where workloads can enjoy hardware acceleration functionality in an operation-level granularity, without direct access to the hardware. To this end, SERRANO introduces vAccel, a virtual acceleration framework tailored for cloud-native, lightweight virtualization setups.

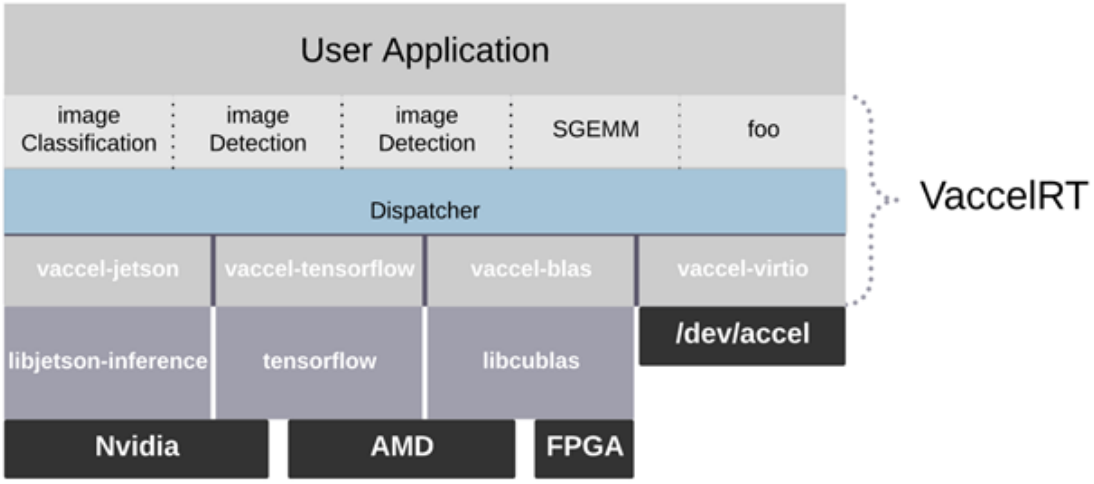


Figure 12: vAccel Framework

vAccel, depicted in Figure 12, is a lightweight framework that exposes hardware acceleration functionality to processes or VMs via a thin runtime system, vAccelRT. vAccelRT abstracts away any hardware/vendor-specific code by employing a modular design where backends

[1] https://5gcomplete.eu/ GA: 871900

implement bindings for popular acceleration frameworks and the frontend exposes a function prototype for each available acceleration function. Using an optimized paravirtual interface, vAccelRT is exposed to a VM's user-space where applications can benefit from hardware acceleration via a simple function call. D4.3 (Section 4) includes a more detailed technical description of these developments.

3.2.4 Lightweight virtualization for seamless deployment

The cloud computing paradigm appears ideal for deploying and managing application execution at scale. However, to support the cloud computing execution model at the edge, devices must support virtualization and run a full hypervisor stack.

Using container technology as an alternative to the full virtualization stack at the edge to increase performance, but keep interoperability, incurs significant security concerns. To this end, SERRANO introduces a hybrid deployment model where lightweight virtualization mechanisms are combined with container deployments to benefit from both worlds. In addition, SERRANO leverages the cloud-native computing paradigm to bring a unified end-to-end deployment experience to the platform users, allowing them to follow a “develop once, deploy anywhere” model.

Specifically, users of the SERRANO platform are able to build a container image using generic tools (Docker, for instance), upload it to a container image registry, and deploy it in edge and cloud resources. This is realized using enhanced container runtime systems that can support VM and container execution, as well as unikernel deployment on a single system.

The system software being developed for the SERRANO platform encompasses the above technologies to form a unified, secure, and efficient software stack that provides both strict security guarantees, while embracing the cloud-native computing paradigm throughout the available resources. Specifically:

- SERRANO enhances current security mechanisms to enforce trusted boot and secure access to data via NBFC's custom hypervisor implementation (KVMM).
- SERRANO employs an ultra-fast workload instantiation mechanism due to lightweight virtualization mechanisms, both available as open-source components (AWS Firecracker) and developed/enhanced as part of the SERRANO software stack (KVMM).
- Using hardware/software co-design mechanisms, SERRANO enables hardware acceleration for tasks running isolated on lightweight VMs, allowing for secure, low-latency responses for isolated multi-tenant compute-intensive tasks running on edge nodes without sacrificing interoperability with cloud interfaces and platforms.
- Engaging in the Serverless computing paradigm, scaling is realized horizontally, while enabling task/workload migration between cloud and edge nodes due to the stateless characteristics of the deployable workloads.

SERRANO completes the systems software stack by employing a unified, end-to-end orchestrator tailored to specific workloads and resource demands, spawning unikernels, containers, or VMs, depending on the tasks' needs. More technical details about these developments are available in D5.3 (Section 5).

4 SERRANO Platform Components

SERRANO adopts a lifecycle methodology (Figure 13) to facilitate application deployment and management. Initially, users will provide their applications along with a high-level infrastructure agnostic (application intent) description of their requirements (*step a*). Next, SERRANO will perform the application profiling and decompose the high-level requirements into specific service goals (*step b*). Then, the allocation of resources to the application's microservices occurs (*step c*) through the cognitive orchestration mechanisms. SERRANO orchestration mechanisms are also responsible for coordinating the application deployment and the efficient movement of required data across the selected resources. Finally, the service assurance mechanisms based on real-time telemetry and appropriate machine reasoning techniques safeguard that applications perform as intended (*step d*) while triggering any required re-optimization.

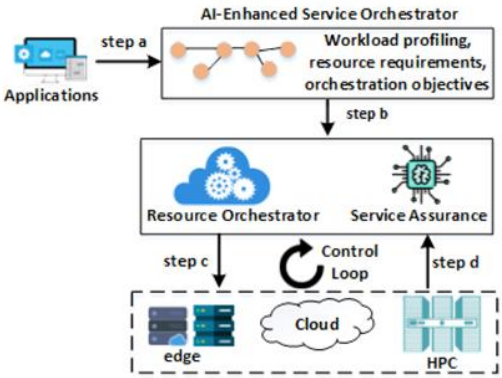


Figure 13: Lifecycle workflow for SERRANO applications

In what follows, we present the SERRANO components.

4.1 AI-enhanced Service Orchestrator

4.1.1 ARDIA Framework

The parameters of interest for executing the UC applications through the SERRANO platform should be specified in advance based on a series of Abstraction Models that that are developed within the SERRANO. The Abstraction Models (analytically described in the deliverable D5.1) define a predefined format so that they can be consumed by the other components of the SERRANO platform. Moreover, since these models define the semantics and structure for the applications', services', and resources' needs in the context of SERRANO, they have also guided the interface specifications of SERRANO components.

For the meaningful interaction among the components of the SERRANO platform, the correspondence among the elements included in the Abstraction Models should be specified so that the UC application requirements can be automatically translated to the appropriate Resource Requirements considering the overall goal of the end user, the possible deployment scenarios, and the collected SERRANO telemetry data.

Table 3: Description of ARDIA Framework Abstraction Model(s)

Component ID	WP5T1AF
Name	ARDIA Framework
High level description	Provides the terminology for the formal description of the requirements of applications, services and resources as well as the profile of the telemetry data monitored and captured by the SERRANO platform.
Collaborators	- UCs provide the high-level requirements of their applications and services/tasks based on the ARDIA modelling framework - AI-enhanced Service Orchestrator uses it for expressing the application and service/task needs in such a form that facilitates their usage by the Resource Orchestrator - Resource Orchestrator uses it for expressing resource requirements - Central Telemetry Handler uses it for expressing the collected infrastructure usage data
UCs	The resources, services, and applications of each use case will be expressed based on the SERRANO abstraction models. The population of the models is part of the UCs implementations and is bound to the telemetry infrastructure capabilities.

Figure 14 presents the internal components of the ARDIA Framework and the interaction of this component with the other entities of the SERRANO platform.

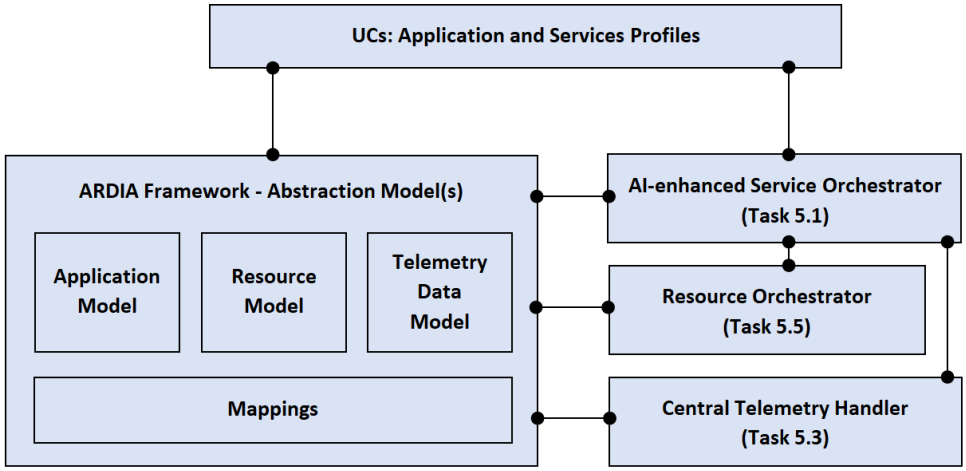


Figure 14: ARDIA Framework components and dependencies

Table 4: Description of Application Model

Component ID	WP5T1AFAM
Name	Application Model
High level description	It provides the necessary elements for the representation of the UCs in terms of their high-level requirements, including intent, design, implementation and deployment information as well as constraints and capabilities.
Collaborators	- The UCs to express the high-level requirements and other metadata of the application to be deployed - AI-enhanced Service Orchestrator for translation (internal) and interfacing with the Telemetry API (e.g., detection of telemetry data about the same or similar applications) as required - Central Telemetry Handler for expressing any application-level information (both for capturing and communication purposes)
UCs	It enables the UC owners to specify their application requirements in a machine-readable manner, so that they can be accordingly used by the SERRANO platform for their successful execution.

Table 5: Description of Resource Model

Component ID	WP5T1AFRM
Name	Resource Model
High level description	It provides the necessary elements for the intermediate or low-level representation of SERRANO (physical and software) resources, in terms of their capabilities (incl. performance, security), utilization, deployment details, and overall, as part of SERRANO infrastructure.
Collaborators	- AI-enhanced Service Orchestrator for expressing UC requirements using the terminology specified in the Resource Model - Resource Orchestrator for expressing the resource requirements to be communicated to the Infrastructure Provider(s) based on the data expressed using the elements of the Resource Model - Central Telemetry Handler for expressing resource related monitoring information
UCs	It provides all the necessary elements for the formal expression of UCs requirements in an intermediate or low-level so that they can be accordingly used for Resource Orchestration purposes.

Table 6: Description of Telemetry Data Model

Component ID	WP5T1AFTM
Name	Telemetry Data Model
High level description	It provides the appropriate elements for representing information about SERRANO infrastructure (from physical and software resources), including their overall status and communication data. This model may include high-level and intermediate or low-level data that will be incorporated in the Application & Resource Models, respectively.
Collaborators	- Service Orchestrator for high-level decisions regarding the services/applications orchestration - Resource Orchestrator for low-level decisions regarding the service/applications execution in each resource

	- Telemetry sub-system for expressing measured parameters
UCs	The Telemetry Data Model also provides all the necessary elements for the formal expression of Telemetry Data from the UCs infrastructures'.

4.1.2 AI-enhanced Service Orchestrator

The AI-enhanced Service Orchestrator facilitates the execution of applications through the SERRANO platform considering the metadata provided about each application, the parameters being necessary for the deployment purposes of the application by the Resource Orchestrator, the status of infrastructure as reported by the Central Telemetry Handler, and the overall goal of the end user (aka intent) at that time. All the data above should be expressed based on the elements of the Abstraction Model(s) of the ARDIA framework presented in Section 4.1.1.

Table 7: Description of AI-enhanced Service Orchestrator

Component ID	WP5T1AISO
Name	AI-enhanced Service Orchestrator
High level description	This component analyses the applications' profiles and translates their high-level requirements into specific infrastructure-related operational constraints and orchestration objectives by considering both the application and services requirements and their forecasted needs.
Collaborators	- UCs for providing the high-level requirements of their applications and services/tasks based on the ARDIA modelling framework - Abstraction Models are used for expressing the application, service and task needs - Resource Orchestrator is provided with the infrastructure related operational constraints and orchestration objectives - Central Telemetry Handler provides the data for the forecasting mechanisms
UCs	It will be fed with the UC application details based on the SERRANO Abstraction Models and will translate their requirements into possible constraints and objectives for the Resource Orchestrator.

Component Architecture

The internal components of the AI-enhanced Service Orchestrator along with the interaction of this component with the other entities of the SERRANO platform are presented in Figure 15.

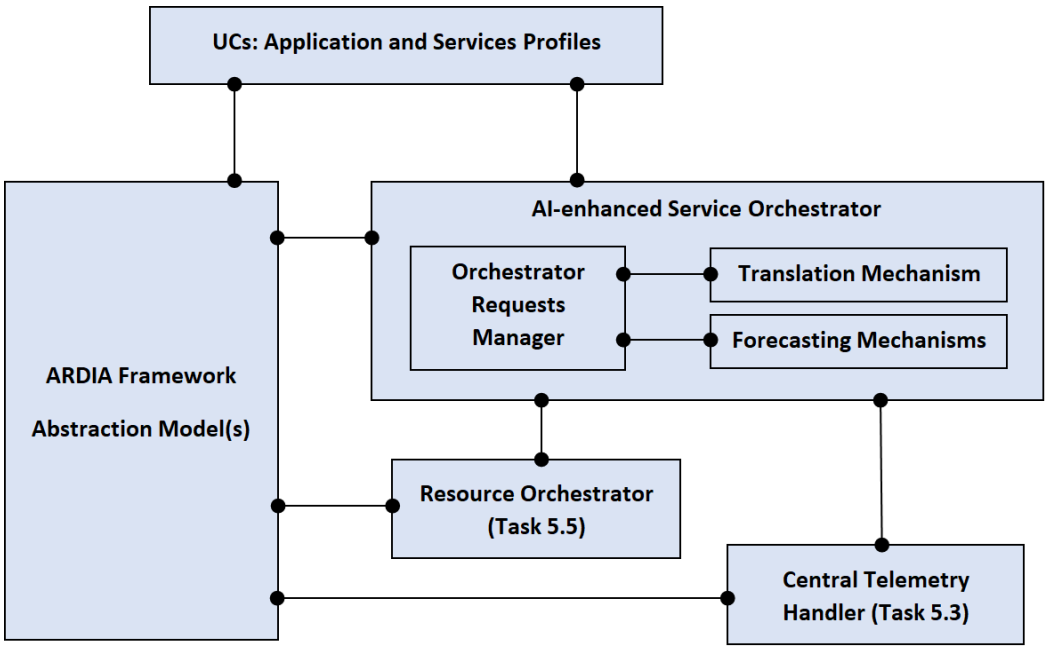


Figure 15: AI-enhanced Service Orchestrator core components and dependencies

Table 8: Description of Orchestrator Requests Manager

Component ID	WP5T1AISOM
Name	Orchestrator Requests Manager
High level description	This component is responsible for managing the requests to the AI-enhanced Service Orchestrator and mediating between the translation and the forecasting mechanisms for synchronizing their interactions.
Collaborators	- Translation Mechanism is being fed with the specified UC application requirements, goals and the forecasted needs of the applications, services and tasks - Forecasting Mechanisms (for AI-enhanced Service Orchestrator) for receiving requests and providing forecasted expected needs for informing the Requests Manager - ARDIA Framework is used for expressing the application, service, task needs
UCs	It will be fed with the UC application details based on the SERRANO Abstraction Models and will coordinate the AI-enhanced Service Orchestrator sub-components for translating their requirements into possible constraints and objectives for the resource orchestrator.

Table 9: Description of Translation Mechanism

Component ID	WP5T1TM
Name	Translation Mechanism
High level description	It analyses the applications profiles and translates their high-level requirements into specific infrastructure related operational constraints and orchestration objectives.
Collaborators	- The ARDIA Framework is used for expressing the application, service, task needs and providing the mappings among the different models - The Orchestrator Requests Manager is triggered to feed the component with the expected needs for adjusting its output
UCs	It will be fed with the UC application details based on the SERRANO Abstraction Models and will translate their requirements into possible constraints and objectives for the resource orchestrator.

Table 10: Description of Forecasting Mechanisms (for Service Orchestrator)

Component ID	WP5T1FM
Name	Forecasting Mechanisms (for Service Orchestrator)
High level description	This component offers the forecasting capabilities to the AI-enhanced Service Orchestrator related to the expected needs of the application, services and tasks.
Collaborators	- The Orchestrator Requests Manager is fed with the forecasting results for the Service Orchestrator to enrich/adjust its output - The Central Telemetry Handler is used for collecting relevant UC application data from the SERRANO infrastructure - The ARDIA Framework is used for expressing the application, service, task needs and telemetry data
UCs	It will be fed with the UC application, services and tasks details and relevant collected telemetry infrastructure data that are formally expressed based on the SERRANO Abstraction Models.

4.2 Resource Orchestrator and Optimization Toolkit

4.2.1 SERRANO resource orchestrator

The realization of SERRANO's ambition requires the efficient and seamless orchestration of the available edge, cloud, and HPC resources. SERRANO is aligned with the current transition from top-down-designed architectures that apply centralized resource control towards federations of loosely coupled autonomous or semi-autonomous systems that are self-organized in a distributed manner. In SERRANO, the overall orchestration is performed in a lean, automated, holistic, and integrated manner, overcoming the complexity barriers stemming from the heterogeneity of computing units.

SERRANO follows a hierarchical architecture to enable end-to-end cognitive resource orchestration and support transparent application deployment over heterogeneous resources. The architecture includes one high-level orchestrator, the Resource Orchestrator, and multiple Local Orchestrators that handle the individual parts of the overall infrastructure. The Resource Orchestrator receives resource and performance requirements from the AI-enhanced Service Orchestrator based on applications the latter serves (SERRANO lifecycle steps a & b) and coordinates their allocation to resources and the initial deployment (step c). In addition, SERRANO Resource Orchestrator adopts a declarative approach, instead of an imperative one, for describing the workload requirements to the selected Local Orchestrators. The selected design approach provides several degrees of freedom to each Local Orchestrator for serving the deployment request optimally, satisfying both the central orchestrator and the resource's objectives.

SERRANO considers that Local Orchestrators are based on existing and well-established solutions. More specifically, the orchestration platform for the edge and cloud platforms is the Kubernetes (K8s) [13], whereas for the HPC platforms HPC resource managers and batch jobs schedulers, such as Slurm [14], and PBS-based (e.g., TORQUE [15], OpenPBS [16]) are considered. Furthermore, for implementing the proposed hierarchical resource orchestration, SERRANO also provides a set of Orchestration Drivers at the resource layer to deal with the low-level details of multi-technology and heterogeneous Local Orchestrators. These drivers provide loose coupling between the central Resource Orchestrator and the underlying resources by expressing the general requests to explicit instructions for the individual Local Orchestrator that then will be responsible for the actual deployment.

Figure 16 shows the updated design of the SERRANO Resource Orchestrator and Orchestration Drivers, the core building blocks and their interactions with other SERRANO components.

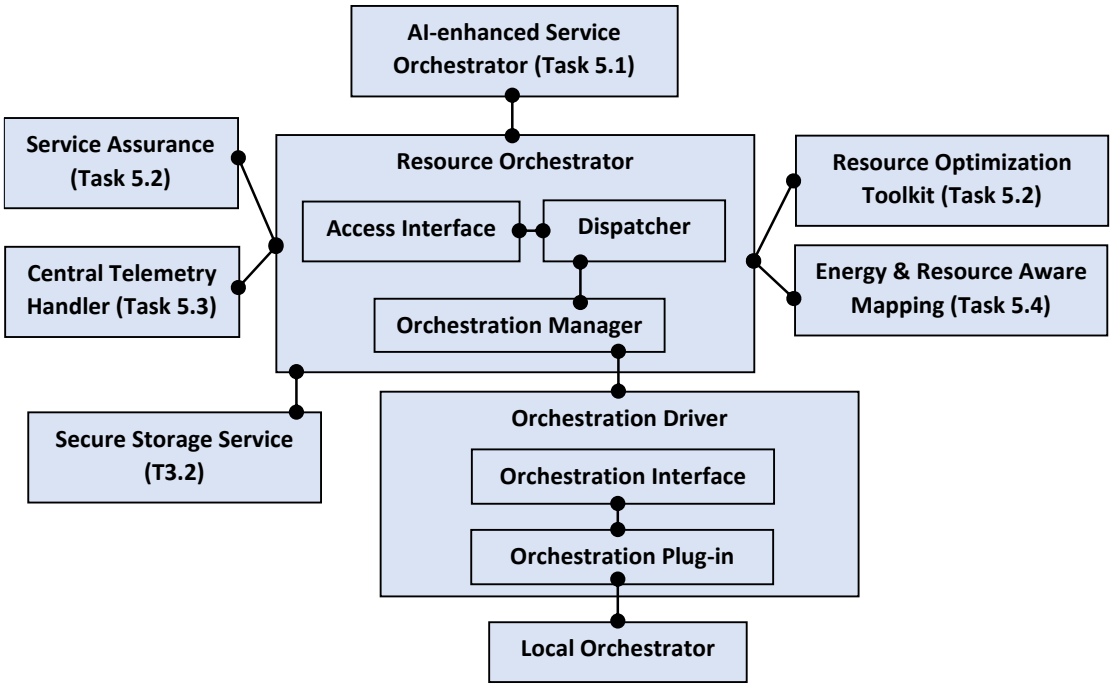


Figure 16: Resource Orchestrator and Orchestration Drivers core components and dependencies

More technical details regarding the initial version of the SERRANO Resource Orchestrator are provided in deliverable D5.3 “Resource orchestration, telemetry and lightweight virtualization mechanisms” (M15). Table 11 provides a high-level description of the core components along with the role of each one. The final version of the Resource Orchestrator and Orchestration Drivers will be reported in D5.4 “Intelligent service and resource orchestration mechanisms” (M31).

Table 11: Description of resource orchestrator and orchestration drivers core components

Component ID	WP5T5RO
Name	Resource Orchestrator
High level description	It ensures efficient service orchestration and resource management in the disaggregated and heterogeneous SERRANO infrastructure. Taking as input from the AI-enhanced Service Orchestrator the application’s high-level requirements and a candidate set of appropriate resource configurations, the Resource Orchestrator, through the Resource Optimization Toolkit and Energy & Resource Aware Mapping components, allocates the proper configuration of hardware and data resources fulfilling the constraints of the individual tasks. It also initiates the application deployment and automatically coordinates the necessary supplemental actions (e.g., transfer of required data).
Collaborators	- ARDIA Framework (WP5T1AF) - AI-enhanced Service Orchestrator (WP5T1AISO) - Resource Optimization Toolkit (WP5T2ROT) - Central Telemetry Handler (WP5T3CTH) - Energy & Resource Aware Mapping (WP5T4EMT) - Orchestration Driver (WP5T5OD) - Service Assurance (WP5T5SA)
UCs	It will coordinate the resource allocation and workload deployment procedures according to AI-enhanced Service Orchestrator instructions derived by analysing the use case preferences.

Component ID	WP5T5AI
Name	Access Interface
High level description	It handles the interaction with the AI-enhanced Service Orchestrator and the other high-level SERRANO components (e.g., Service Assurance). It acts as the single-entry point for the other components to SERRANO resource orchestration functionalities. Initially, the Access Interface will validate the requests and then forward them to the Dispatcher.
Collaborators	- ARDIA Framework (WP5T1AF) - AI-enhanced Service Orchestrator (WP5T1AISO) - Dispatcher (WP5T5D) - Orchestration Manager (WP5T5OM) - Service Assurance (WP5T5SA)

UCs	It will trigger the necessary internal procedures for the appropriate resource allocation and initial deployment of workloads based on UCs preferences.

Component ID	WP5T5D
Name	Dispatcher
High level description	It implements the application logic, oversees the operation of the other internal components and coordinates the resource allocation and application deployment operations. It also interacts with the ROT to retrieve the application deployment instructions.
Collaborators	- ARDIA Framework (WP5T1AF) - Access Interface (WP5T5AI) - Resource Optimization Toolkit (WP5T2ROT) - Energy & Resource Aware Mapping (WP5T4EMT) - Orchestration Manager (WP5T5OD)
UCs	It will trigger the necessary internal procedures for the appropriate resource allocation and initial deployment of UCs workloads based on their preferences.

Component ID	WP5T5OM
Name	Orchestration Manager
High level description	It prepares the necessary application deployment instructions based on the Resource Optimization Toolkit's (ROT) decisions and the characteristics of the selected Local Orchestrators. Moreover, it coordinates the required data movement through the interaction with the Secure Storage Service and triggers the application deployment at the selected edge/cloud/HPC platforms.
Collaborators	- Dispatcher (WP5T5D) - Secure Storage Service - Orchestration Driver (WP5T5OD)
UCs	It will setup and coordinate the application deployment at the selected individual edge/cloud/HPC platforms.

Component ID	WP5T5OD
Name	Orchestration Driver
High level description	It provides an abstraction layer for interacting with the specific edge, cloud, and HPC orchestration mechanisms at each SERRANO location. It receives by the Resource Orchestrator requests for deploying or adjusting already deployed applications.
Collaborators	- Orchestration Manager (WP5T5OM) - Orchestration Interface (WP5T5OI)

UCs	It will handle the requests from the SERRANO Resource Orchestrator regarding the initial deployment and re-optimization of submitted applications.

Component ID	WP5T5OI
Name	Orchestration Interface
High level description	It provides an infrastructure agnostic interface for describing the deployment preferences and constraints to the heterogeneous local orchestration mechanisms. It also handles the interaction with the SERRANO Secure Storage Service.
Collaborators	- Orchestration Manager (WP5T5OM) - Orchestration Plug-ins (WP5T5OP) - Secure Storage Service
UCs	It will facilitate the transparent deployment of submitted applications at heterogeneous SERRANO infrastructure.

Component ID	WP5T5OP
Name	Orchestration Plug-ins
High level description	This component translates the generic instructions from the Orchestration Interface to specific actions and procedures for the selected local orchestration mechanisms. The interaction is based on the API and tools exposed by each local orchestration component. There is available a specific plug-in for each supported local orchestration mechanism.
Collaborators	- Orchestration Interface (WP5T5OI) - SERRANO Lightweight Virtualization Mechanisms - Local Orchestrator
UCs	It will handle the actual transparent deployment and execution of submitted applications at heterogeneous SERRANO infrastructure.

4.2.2 Resource optimization toolkit

The complexity and heterogeneity of distributed computing infrastructures pose significant management and service deployment challenges. Heading to more complex environments, the development of intelligent orchestration algorithms is key to optimize resource utilization for present and future workloads. SERRANO will exploit multi-objective optimization, graph theory, AI/ML techniques, and heuristics to design a set of algorithms aiming to provide different trade-offs between optimality and complexity. The Resource Optimization Toolkit (ROT) will integrate these algorithms in the SERRANO platform, implementing the decide part at the envisioned closed-loop control based on the principles of observe, decide, and act.

The ROT provides the SERRANO Resource Orchestrator with the required logic to allocate the edge, cloud, and HPC resources to satisfy the applications' requirements, coordinate the

efficient movement of required data across the selected resources and support proactively and reactively re-optimization adjustments.

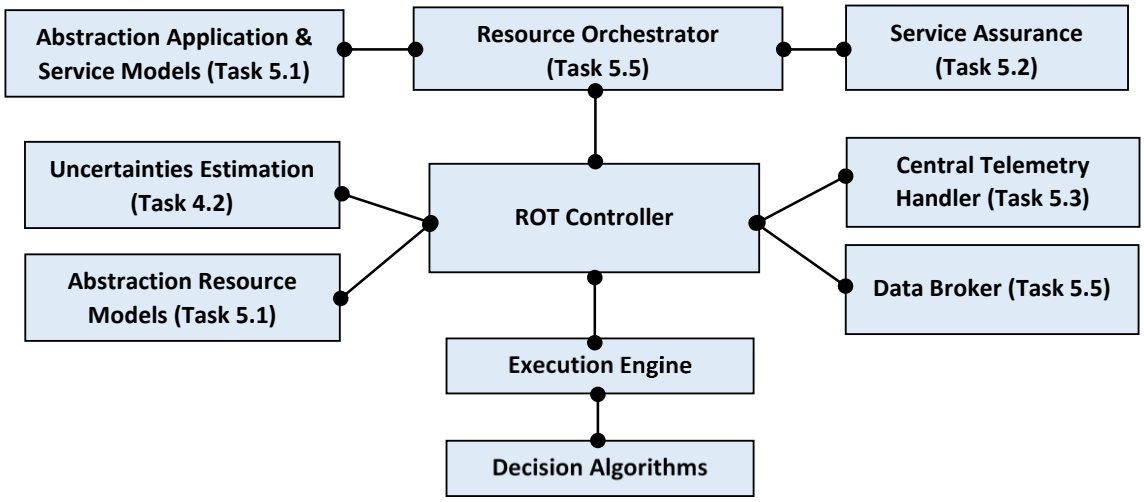


Figure 17: Resource Optimization Toolkit core components and dependencies

Task 5.2 is responsible for developing the resource allocation algorithms and the design and implementation of the ROT. It is designed to provide fast execution and efficient resource usage while scaling with the demands. According to the selected design, there is one Controller and multiple workers. Each worker is composed of the Execution Engine and the library of the decision algorithms. Additional information for the initial version of the ROT is provided in deliverable D5.2 "Algorithmic framework, performance and power models" (M15). Figure 17 presents the updated version of ROT's key building blocks and the interactions with other SERRANO components. Table 12 provides a high-level description of these components.

Table 12: Description of resource optimization toolkit core components

Component ID	WP5T2ROT
Name	Resource Optimization Toolkit
High level description	It provides the implementation of the developed multi-objective optimization and orchestration algorithms. In addition, it is responsible for preparing, coordinating, and managing the appropriate algorithm's execution to facilitate SERRANO Resource Orchestrator to its application deployment and service assurance functionalities.
Collaborators	- ARDIA Framework (WP5T1AF) - Resource Orchestrator, the component that triggers the execution of some specific algorithm (WP5T5RO) - Central Telemetry Handler, provides details for the current state of resources (WP5T3CTH) - Resource Abstraction Models, provide details for the resource requirements (WP5T1AM)
UCs	It will provide the required high-level orchestration decisions to SERRANO Resource Orchestrator to deploy and re-optimize the UC applications' workloads.

Component ID	WP5T2RT
Name	ROT Controller
High level description	It receives execution requests from the Resource Orchestrator and handles the interaction with the multiple instances of the Execution Engine. It also interacts with the Central Telemetry Handler to retrieve the required information.
Collaborators	- Resource Orchestrator, the component that triggers the execution of some specific algorithm (WP5T5RO) - Central Telemetry Handler, provides details for the current state of resources (WP5T3CTH) - Data Broker, asynchronous communication between the ROT Controller and Execution Engines (WP5T5MB) - Execution Engines (WP5T2EE)
UCs	It will provide the required high-level orchestration decisions to SERRANO Resource Orchestrator to deploy and re-optimize the UC applications' workloads.

Component ID	WP5T2EE
Name	Execution Engine
High level description	It receives requests, for starting or terminating algorithm executions, through the predefined interface from the ROT Controller and performs all the required actions, including preparing the execution environment, monitoring progress, and forwarding the results to Controller.
Collaborators	- Data Broker, asynchronous communication between the ROT Controller and Execution Engines (WP5T5MB) - Decision Algorithms (WP5T2DA)
UCs	It will provide the execution of the decision algorithms within the SERRANO platform.

Component ID	WP5T2DA
Name	Decision Algorithms
High level description	The library of multi-objective optimization and orchestration algorithms. The integrated algorithms will achieve different trade-offs between optimality and complexity to efficiently satisfy the heterogeneous and strict applications' requirements.
Collaborators	- Execution Engine (WP5T2EE)
UCs	It will provide the required high-level orchestration decisions to SERRANO Resource Orchestrator to deploy and re-optimize the applications' workloads.

4.2.3 Energy and resource aware mapping

The SERRANO Resource Orchestrator will allocate computational resources fulfilling certain criteria. One of these criteria is energy efficiency. Hence, the energy-aware design, configuration, and execution of the UC applications must be supported by the SERRANO platform. This relates mainly to the provided HPC services.

If an application meets the requirements for running in an HPC environment, such as high-level degree of parallelization and problem size (see section 5.2.1.2 SERRANO HPC Resources in D2.2 for more details), the resource orchestrator will decide whether to execute the application or part of it in the cloud or on an HPC platform.

This decision depends on certain factors and their relation. Here are the most important ones:

- User preferences;
- Amount of input-output data;
- Amount of computation work and data transfers between various levels of the memory hierarchy;
- Amount of inter communication;
- Performance and energy efficiency of the present hardware.

SERRANO aims to develop a framework that will assist developers in incorporating the performance and power model functionality into the design, programming, and execution of the HPC services, particularly multi-dimensional FFT, Kalmar filter, and others. This is essential because performance optimization and the optimal execution configuration of an HPC application are crucial for energy efficiency.

The framework will support the performance optimization of the individual phases of the HPC services, the so-called kernels, such as IO, memory-bound, and computational-bound phases. Moreover, it will also provide information about their behaviour depending on the hardware and their configurations, such as the number of required compute nodes, number of processes and threads on one node, and CPU frequency. Next, we present its main components and the interactions with other SERRANO components.

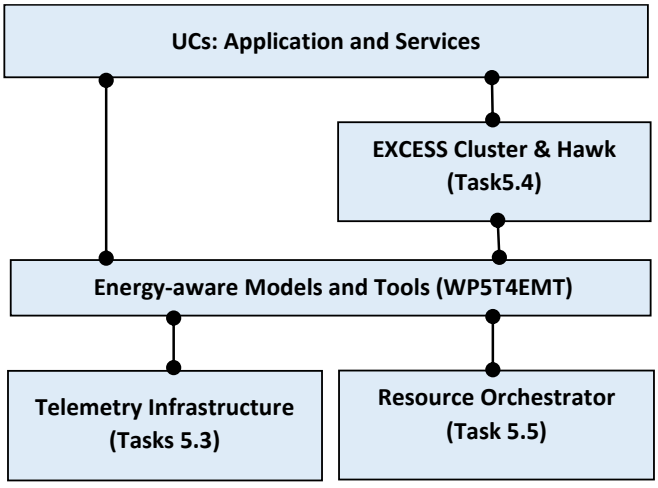


Figure 18: Energy and resource aware mapping core components and dependencies

Table 13: Description of energy and resource aware mapping core components

Component ID	WP5T4EXCESS
Name	EXCESS Test Cluster
High level description	It is one of the main components in integrating the functionality of power and energy models into the design and programming models of digital services, particularly on HPC systems as part of the SERRANO platform. The power measurement system and a set of power and energy analysis tools provide information about the features of user applications to optimize their implementation and execution. Different hardware and software components such as operating and runtime systems should be considered as far as possible.
Collaborators	A set of energy-aware benchmarks will be selected based on the analysis performed in WP2 (Task 2.1 and Task 2.2) and WP3 (Task 3.1), which will be used to identify and compare the characteristics (strengths and weaknesses) of the conventionally HPC (e.g., Supercomputer "Hawk" and EXCESS cluster) and of the SERRANO related platforms (see Task 4.1).
UCs	It will consider the most intensive computational kernels of UC applications. Hence, it will contribute to the workflow optimization of UC applications in the SERRANO platform.

Component ID	WP5T4EMT
Name	Energy-aware Models and Tools
High level description	Based on the previous step's results (see description of component WP5T4EXCESS), we will extend the existing power and performance models, adapting them to the specific hardware features (NUMA topology, memory/cache bandwidth, and latencies, CPU-frequencies, network parameters). The proposed models will enable developers to observe the execution of applications and services schematically in particular data flows through system components.
Collaborators	The biggest challenge is to meet the requirements of both HPC and cloud and edge systems. Therefore, compatibility with SERRANO components is necessary (WP5T3CTH).
UCs	HPC services, based on the use cases requirements, need a framework for efficient execution of a set of concurrent tasks (see D2.2 section 5.2.5 Requirements for efficient use of supercomputers). The models will help identify the optimal strategies for executing the HPC services on a supercomputer (e.g., Hawk see D2.2 section 5.2.1 “HPC Resources at HLRS”).

4.3 Service Assurance Mechanisms

The management of Quality-of-Service (QoS) guarantees and Service-Level-Agreements (SLAs) in the Edge-to-Cloud computing continuum forms an overly complex task due to the significant heterogeneity of the hardware infrastructure spread across this spectrum, the varying runtime conditions, and unpredictability in terms of client requests. In addition, the need for more cost-efficient infrastructures forms multi-tenancy as a first-class system design concern, which further incommodes the guarantee of high-quality services. As described in more detail in D2.3 (Section 4.3), SERRANO provides several service assurance mechanisms for satisfying the QoS requirements of applications deployed on the platform. Specifically, SERRANO aims to provide such mechanisms across various levels, ranging from hardware-aware tuning techniques up to application-specific optimizations. Below, we explain how these mechanisms have been upgraded/modified compared to the description provided in D2.3.

4.3.1 Plug&Chip extensions for device-aware application mapping on GPU and FPGA accelerators

In its final version, SERRANO provides and exposes a set of services that automatically optimize the source code of applications deployed on top of the platform with respect to the underlying hardware infrastructure, both for FPGA and GPU devices. As described in deliverables D2.3 and D4.3 “Framework for seamlessly integration of heterogeneous workload-aware performance improvement” (M15), these services are implemented as extension plugins to the Plug&Chip framework and are exposed as socket interfaces to be consumed by end-users. Moreover, as described in deliverable D6.3 “The SERRANO integrated platform” (M18), a python library interface has been developed that facilitates the employment of these optimization frameworks by end-users. Below, we briefly describe the extensions developed within the project, while more details are available in the respective technical deliverables of WP4.

4.3.1.1 Plug&Chip extension for fine-tuned GPU acceleration

The Plug&Chip extension for fine-tuned GPU acceleration is a tool that automatically optimizes CUDA applications for different GPU devices in terms of performance. More specifically, AUTH’s work targets the auto-tuning of CUDA kernels by applying a block coarsening transformation (please refer to deliverables D2.3 and D4.3 for more information) with respect to the kernel itself, its input dataset size and the underlying HW architecture. Specifically, a ML-driven approach is adopted, where the prediction target is the estimated latency of a transformed kernel given the underlying GPU architecture and its input dataset size. The machine learning based proposed tool consists of an in-house built source-to-source compiler and a regression model (based on the Extreme Gradient Boosting – XGBoost) that can address the tuning problem across different GPU architectures. More technical details can be found in deliverable D4.3.

4.3.1.2 Code optimizations for fine-tuned FPGA accelerated applications

AUTH is developing an extension for the Plug&Chip framework that automatically optimizes High Level Synthesis (HLS) kernels for FPGA devices leveraging Machine Learning (ML) techniques. The proposed methodology takes as input the synthesizable source code of a C/C++ kernel and provides the set of optimal kernels with added HLS directives, optimized for a specific device and target clock frequency. The core component of this optimization system is the Genetic Algorithm (GA) that generates different HLS configurations (i.e., directives for each action point), and thus efficiently traverses through the immense decision design space. These configurations are applied to the source code of the C/C++ kernel using a Source-to-Source (S2S) compiler. The output of the S2S compiler as well as the device and the target clock frequency are passed to Vitis HLS that provides an estimation of the latency, the utilization percentage of the Block Random Access Memories (BRAM), the Flip Flops (FF), the Lookup Tables (LUT) and the Digital Signal Processing (DSP) units. These estimations are acquired by the GA and are used to create the next set of configurations to be examined. More information concerning the proposed methodology can be found in deliverable D4.3.

4.3.2 DMM4FPGA framework extensions for dynamic memory management of FPGA accelerated kernels

A semi-automated methodology for minimizing the fragmentation of the on-chip memories has been designed along with a garbage collector implemented on the FPGA for performing memory compaction and maximizing memory efficiency. This flow is illustrated in Figure 19.

In the offline analysis phase, the designer, based on the accelerators that will be executed in parallel, performs a Monte-Carlo (MC) analysis to statistically determine the probability of memory allocation failures due to fragmentation. The goal of this analysis is to determine the design parameters that will used in the synthesized design.

Once the offline analysis is completed, the proposed defragmentation mechanism is synthesized and implemented on the FPGA with the selected parameters. During the execution time, the garbage collector is automatically executed when the on-chip memories are fragmented. For more details and evaluation results, please refer to D4.3.

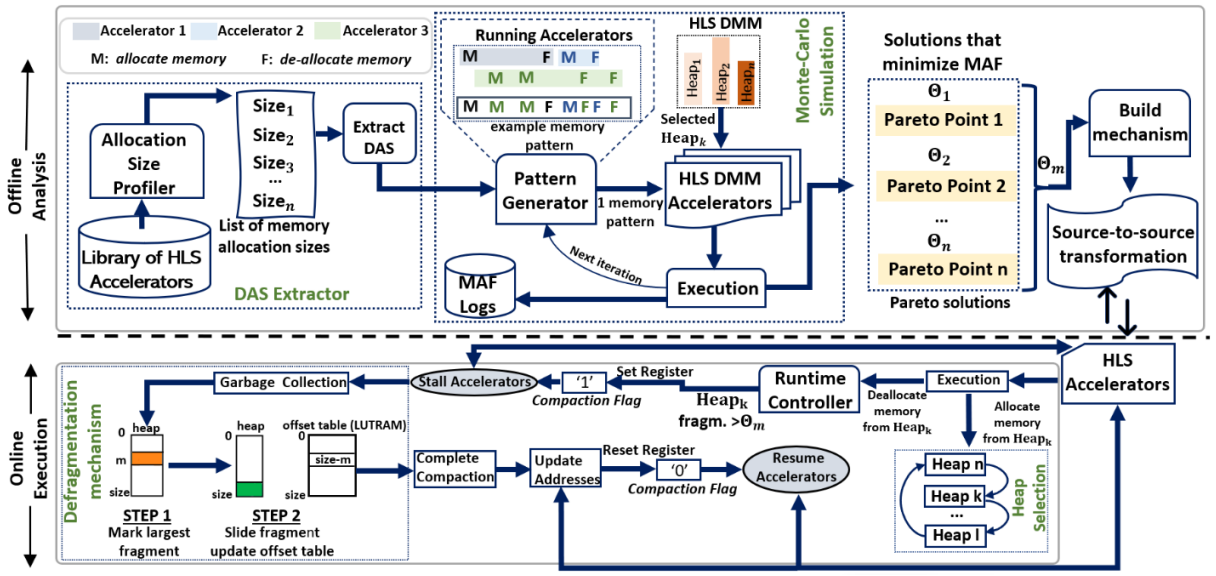


Figure 19: Methodology for minimizing fragmentation of the on-chip memories

4.3.3 Variable-accuracy optimizations based on approximate computing

The SERRANO platform will leverage the REMAP framework to provide runtime QoS guarantees under different stressing scenarios. The REMAP framework enables the modelling and design of variable-accuracy optimizations based on approximate computing. It will be used to support workload-aware runtime adaption among approximate kernels to significantly improve efficiency (hardware utilization and performance per watt metric) without affecting the application's/service's QoS. By utilizing the optimization techniques described in Sections 4.3.1 and 4.3.2, the REMAP framework will interface with the local orchestrator to apply dynamic and input-driven optimizations and approximations by alternating between different versions of accelerated kernels based on telemetry data in a feedback-based approach, i.e., continuously monitor applications' performance and energy-efficiency to drive the run-time decisions of the local orchestrator.

4.3.4 Performance Maximization under Maximum Affordable Error

On the SERRANO platform, the requirements for the accuracy of the input and output data as well as the intermediate calculations are clearly differentiated. The choice of accuracy can vary significantly from one use case to another and must be closely analysed not to produce erroneous results. Additionally, the input and output data format must also be closely considered.

The goal is to reduce the amount of data transferred between the SERRANO platform components and allow users to reduce the computational complexity of their applications. The data interpolation also reduces energy costs because the local computations are up to several orders more efficiently than the data transfer over the network or IO devices. If an application meets the requirements for HPC deployment (see section 3.1.3), its data (e.g., the

signal data) will be filtered and interpolated by the HPC services of the SERRANO platform. Relevant application examples are the Kalman and Fast Fourier transform filters.

The Kalman filter, also called linear quadratic estimation, can estimate unknown variables from a series of measurements that contain statistical noise and uncertainties. As a result, it has a wide range of applications in guidance, navigation, and finance. For example, when an airplane at high-speed shakes due to turbulence, its sensors also shake. Therefore, they do not be able to provide accurate measurements. In this case, the Kalman filter can provide optimal sensor reading from noisy measurements. The filter has two major parameters, R and K, which regulate how much noise will be reduced from the system. After filtering, the signal data can be further processed instead of transferring the data directly to the user for further local analysis, e. g. for anomalies detection. Different techniques can be used depending on the user application's goals. For example, if the signal analysis is simple (e.g., detection of exceeding a certain threshold), it can be applied immediately to the filtered data. Hence, only the individual threshold-events details will be saved for further local analysis.

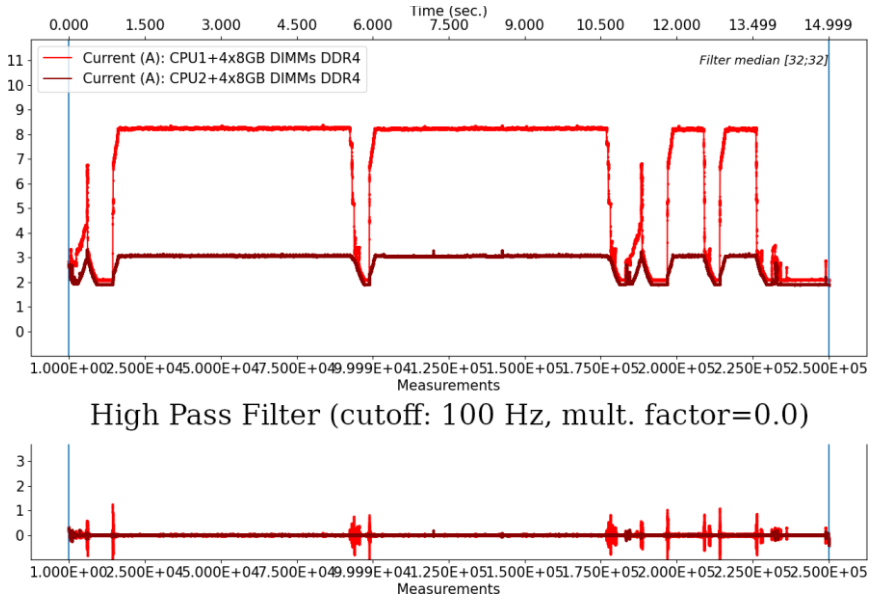


Figure 20: Signal from sensors for measuring the power consumption of two processors (measurement with 16.6 KHz) and the results of applying on it "high pass" FFT filter.

Furthermore, the discrete FFT maps a signal from its original domain to the domain of the selected frequencies and vice versa. Thus, it is used, among the others, to analyse data for irregularities. Figure 20 shows the signal from the sensors for measuring the power consumption of two processors during the execution of an application on one of them. The measurement was done with 16.6 KHz. The diagram on the top shows the results after applying the simple median filter. Several computational phases are clearly seen in the figure. The results of the "high pass" FFT filter are shown in the diagram at the bottom. The same phases of the application can be identified more easily and require fewer data to transmit filtering results.

The appropriate fine-tuning of the formats used for the data representation provides one additional option to reduce the required data transfers. The native data formats implemented in the HPC hardware are integer (4, 8 bytes) and floating-point numbers (4, 8 bytes)[2]. The HPC services will use them appropriately, based on the application requirements and the SERRANO Resource Orchestrator specifications. Other formats, such as the ASCII format, that are used for saving data in comma-separated values (CSV) tables, must be avoided in HPC platforms since they impose additional costs for the following reasons:

- Representing numbers in the native data format requires a constant number of bytes, such as four bytes for single-precision floating-point numbers of type real32 or eight bytes for type real64. In ASCII format, the numbers are represented as fixed-point numbers. Therefore, a maximum of four digits of a number can be stored in four bytes. However, in most cases, the four or even eight digits are not enough to represent the signal data without significant signal deformation.
- When using the ASCII format, the data must be converted to the format supported by the hardware. This operation costs time and energy and, as in the case of IO, can drastically reduce the performance. It also complicates the parallel reading and writing of the files with the data because the field lengths in the data can vary.

SERRANO develops the HPC services for filtering the signal data along with the development of the VVUQ framework to determine the optimal filter parameters and data formats for the particular use cases. Furthermore, the boundary conditions for the application of these HPC services will be determined depending on the particular parameters, such as the required bandwidth of data transmission and minimum data size. We are currently developing the framework for the Kalmar filter as a hybrid OMP/MPI application that can efficiently check different parameters in parallel on the user data and report on the differences.

4.3.5 Service assurance and remediation

To successfully implement an execution plan that ensures a certain level of quality in an application lifecycle (from the involved services and requested resources to performance and costs mitigation), we need to build a loosely coupled model that implements at least the following components: (a) planning, (b) implementation, (c) diagnosis and (c) remediation.

The **planning component** will generate the deployment scenarios based on the application profile and the additional resource needs of the respective application. The **implementation component** will select the most appropriate deployment scenario, orchestrate the available workloads by building the corresponding implementation plan, and launch their deployment in the selected resources. Also, the monitoring system, capable of gathering data from all the resources or applications, must be setup and ensure that all the monitoring data is collected, processed, and made available for other components to consume. The **diagnosis component** will consume the monitoring data and analyse and classify the collected monitoring events to

[2] The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is most widely used format for floating-point arithmetic. We do not consider here hardware such as GPUs and FPGAs and thus half and customs precision formats, since these are not supported by the hardware of the considered HPC resources.

identify anomalies among the resources with respect to the implementation plan. Finally, the **remediation component** will dispose of contra-measures, based on the identified anomalies, to reinstate the execution scenario to the original state or inform the corresponding orchestration components about a flagrant execution plan specifications violation. In the latter case, the execution scenario needs to be reconfigured to meet the expectations derived from the execution plan specifications.

The planning component corresponds to the AI-Enhanced Service Orchestrator (Section 4.1) and the implementation component to the SERRANO Resource Orchestrator (Section 4.2). The diagnosis and remediation components are the key building blocks of the SERRANO Service Assurance mechanisms at the central and local levels and consist of the following components:

- Data ingestion – responsible for data retrieval from different external sources using different transfer protocols.
- Data pre-processing – the ingested data needs to be altered to fit the requirements of the event detection engine (formatting, statistical analysis, etc.).
- Event detection engine – a complex component that makes use of ML mechanisms to detect anomalous events based on the application life-cycle specification.
- Data bus – exposes the internal repositories (trained and/or validated models, pre-processed data etc.) to other internal or external components (i.e., local service assurance instances); it is also used for external interaction with the service assurance system by allowing external entities to push specific data for internal use.
- Remediation – triggers remediation plans or informs orchestration components about the identified violations of the execution plans' requirements or specifications.

The orchestration mechanisms will automatically configure the diagnosis component to analyse and detect anomalous events using the telemetry data collected by the SERRANO Cloud and Network Telemetry components. The remediation component will prepare a remediation plan based on the diagnosis report where the detected anomalies are outlined. It will implement a Belief-Desire-Intention (BDI) model. The beliefs represent the metrics (i.e., type, value, minimum and maximum, frequency, or other measurable conditions) that describe a component specification. The desires represent the targeted quality assurance policies based on the component's specific metrics. The intentions are the remediation actions that define how the service assurance component should react to execution violations. The event detection engine can also be used by applications that need to detect anomalies, independent of the service assurance and remediation system.

Figure 21 shows the main components of the service assurance and remediation mechanisms and Table 14 provides their description.

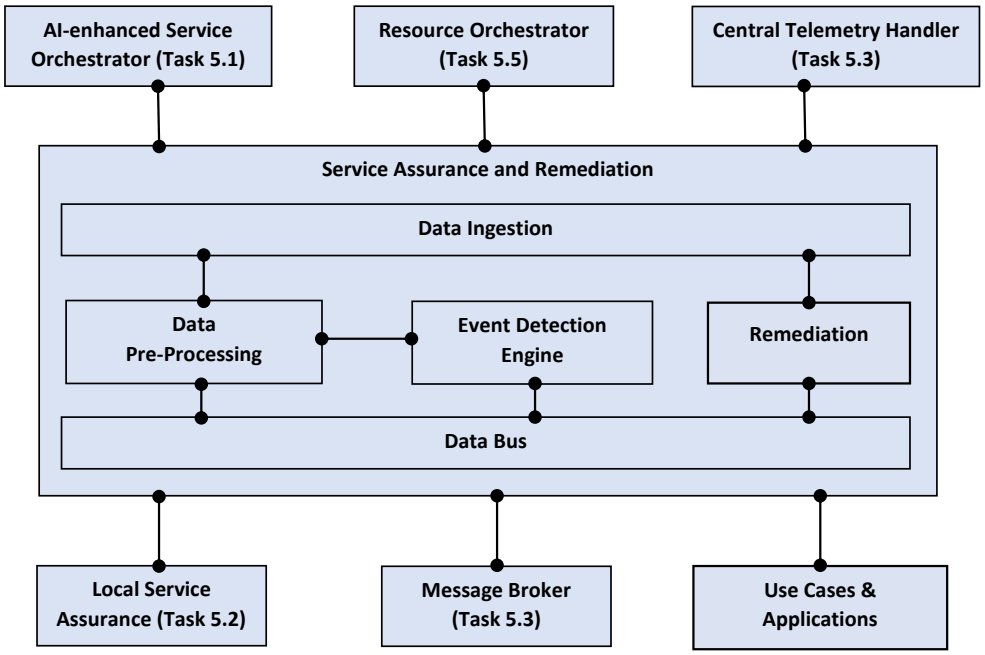


Figure 21: Service assurance and remediation system core components and dependencies

Table 14: Description of service assurance and remediation system core components

Component ID	WP5T5SA
Name	Service Assurance and Remediation
High level description	The service assurance and remediation mechanisms will ensure that the deployed applications are executed within the normal boundaries set by the requirements and constraints defined by the users. It will be designed as a distributed system with multiple instances, at the central and local levels, that coexist and cooperate to trigger pro-actively and re-actively dynamic adjustments.
Collaborators	- Resource Orchestrator (WP5T5RO) - Central Telemetry Handler (WP5T3CTH) - Resource Optimization Toolkit (WP5T2ROT) - Message Broker (WP5T5MB) - UCs or applications that needs event detection services
UCs	It will ensure that the applications perform as intended, while it will dynamically trigger the necessary adjustments if the current deployments do not comply with the requested SLA guarantees.

Component ID	WP5T5EDE
Name	Event Detection Engine
High level description	The event detection engine (EDE) can detect complex anomalous events both from the SERRANO platform and the target UCs. The causes for these anomalous events can range from hardware issues, misconfigurations up to application bugs. Because EDE is application agnostic, it only requires access to telemetry data for the applications. Moreover, EDE includes the necessary tools and interfaces that enable

	external entities to setup the detection methods (based on supervised and unsupervised methods). The two main subcomponents handle training and prediction jobs. The training subcomponent enables optimization methodologies geared towards transprecise adaptation, meaning that a computationally optimized version of the detection models can be deployed on local service assurance instances. Once the prediction subcomponent detects anomalies, they are then reported using the SERRANO Data Broker via Kafka topics. The reporting mechanisms will add root cause analysis metadata to all detected anomalous instances. These can include feature importance data, Game-theoretic explanations (i.e., Shapely values), decision boundaries, etc. This way, other tools from the SERRANO toolchain can use the detected anomalies and their probable causes to further optimizations (i.e., Resource Orchestrator, Resource Optimization Toolkit).
Collaborators	- Resource Orchestrator (WP5T5RO) - Service Assurance Data Pre-processing (WP5T5SADP) - Service Assurance Data Bus (WP5T5SADB)
UCs	It will provide detection capabilities in case of anomalous events, including but not limited to hardware resources, performance, applications. The resulting anomaly report will be made available via the SERRANO Message Broker to other tools in the SERRANO toolchain. These anomaly reports will be used by the orchestration and optimization solutions to enrich the raw telemetry data thus giving greater insight.

Component ID	WP5T5SADI
Name	Service Assurance Data Ingestion
High level description	A component specialised in fetching monitoring data from various sources (heterogeneous systems and technologies). A specialized source connect is implemented for each specific data source to serve as an adapter for every supported monitoring and/or metrics storage solution. It can load data directly from static files (various formats supported: HDF5, CSV, JSON, or raw). Moreover, the data ingestion component can fetch data directly by querying the monitoring solution (using a predefined interface) or consuming a stream directly from a queuing service. This approach will reduce the time between an event occurrence and the possible anomalous event detection.
Collaborators	- Data Collector (WP5T3DC) - Message Broker (WP5T5MB) - Stream Handler (WP5T5SH)
UCs	It will provide connector for various monitoring data sources to consume valuable information necessary for the other Service Assurance components.

Component ID	WP5T5SADP
Name	Service Assurance Data Pre-processing
High level description	It will apply several transformations to the data provided by the data ingestion component. More specifically it will perform: • data formatting (i.e., one-hot encoding) • statistical analysis • data splitting into training and validation sets • data augmentation (i.e., oversampling and under sampling)
Collaborators	- Service Assurance Data Ingestion (WP5T5SADI) - Event Detection Engine (WP5T5EDE)
UCs	It will analyse and modify the data according to the needs of the event detection analytic components. The resulting datasets will be feed into the analytic engine for preparing and conduct the training and/or prediction operations.

Component ID	WP5T5SADB
Name	Service Assurance Data Bus
High level description	The data bus ensures that the trained and validated models, generated by the event detection engine, and/or pre-processed data are stored, and then made available to be instantiated and used in a production environment. It is also used as a messaging interface with the external components that must interact with the service assurance component. Thus, the bus represents a distributed gateway for accessing several repositories where the models are stored.
Collaborators	- Local Service Assurance - Event Detection Engine (WP5T5EDE) - External components that need to interact with the service assurance component
UCs	It distributes the training and validation models for further use in a distributed production environment. It will also allow the interaction of external components with the service assurance services.

Component ID	WP5T5SAR
Name	Service Assurance Remediation
High level description	The remediation component will generate a remediation plan used by the Resource Orchestrator to bring the anomalous component to nominal parameters. This component will follow a BDI model, where based on a set of beliefs and desires, the system will emit an intention plan, which represents the remediation actions that will trigger the process of reinstating the execution plan to the normal course.
Collaborators	- AI-enhanced Service Orchestrator (WP5T1AISO) - Resource Orchestrator (WP5T5RO) - Orchestration Drivers (WP5T5OD)
UCs	Based on the execution plan specifications (services involved, resources deployed, metrics to monitor, etc.) the remediation component will generate a list of remediation actions to follow in case

	of violations of the execution plan specifications provided by the UCs. This will also include potential remediations actions.

4.4 Cloud and Network Telemetry

In the SERRANO platform, a sense (detect what is happening), discern (interpret senses), infer (understand implications), decide (choose a course of action) and act (take action) continuous control loop will run autonomously over the infinite to adjust resources and deployed applications proactively or reactively. An autonomous, scalable, and data-driven network and cloud telemetry framework will collect and analyse the required telemetry data across the distributed heterogeneous (edge/cloud/HPC) SERRANO infrastructure, providing the sense and discern operations in the envisioned closed-loop control.

Moreover, the appropriate AI/ML methods will be used to enable, among others, the dynamic adaptation of the telemetry infrastructure and the correlation of telemetry information in time and space, to infer appropriate metrics, identify general performance problems, predict the future infrastructure state, and localise failures.

SERRANO leverages different categories of resources to collect the necessary monitoring information. The first category includes the computing and storage resources in the heterogeneous SERRANO infrastructure. The appropriate telemetry data will be collected by the typical resources and the SERRANO-enhanced hardware. Moreover, SERRANO will capitalize on the monitoring capabilities of SmartNICs placed at critical positions in the infrastructure to obtain important network-related telemetry information. The third category includes all the SERRANO-enhanced software level resources. Finally, all the deployed cloud-native applications and short-lived (serverless) services will be automatically monitored.

SERRANO targets a hierarchical telemetry infrastructure to monitor all the above categories of resources intelligently and dynamically, with three key building blocks:

- *Central Telemetry Handler:* the root component in SERRANO hierarchical telemetry infrastructure.
- *Enhanced Telemetry Agents*: a set of generic monitoring entities that each coordinates the operation of a specific set of Monitoring Probes. Additionally, they pre-process, aggregate, and correlate the collected telemetry data.
- *Monitoring Probes*: a set of probes (each one responsible for a specific resource type) that collect the actual telemetry data. Some probes are out of the shelf, while others are implemented in the context of SERRANO.

Figure 22 shows the updated design of the cloud and network telemetry infrastructure, the key building blocks of the, their main components and the interactions with other core components within the SERRANO platform.

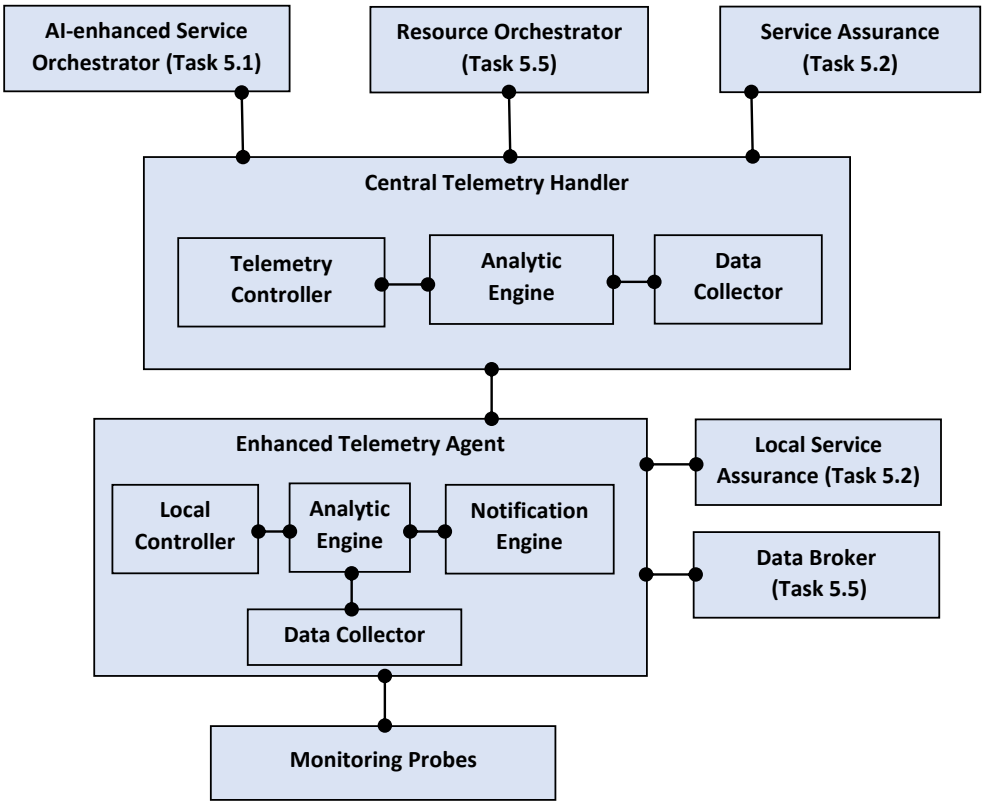


Figure 22: Cloud and network telemetry core components and dependencies

The detailed design and implementation of the cloud and network telemetry within SERRANO is the primary focus of Task 5.3. A more technical presentation of the initial version of the SERRANO telemetry framework is provided as part of deliverable D5.3 "Resource orchestration, telemetry and lightweight virtualization mechanisms" (M15). The following tables provide a high-level description of its core components along with the role of each one. The final version of the telemetry framework will be reported in D5.4 "Intelligent service and resource orchestration mechanisms" (M31).

Table 15: Description of cloud and network telemetry core components

Component ID	WP5T3CTH
Name	Central Telemetry Handler
High level description	It is the root component in SERRANO hierarchical cloud and network telemetry infrastructure. It maintains a global view of the current state of both the infrastructure and the deployed applications. It also coordinates the overall operation of the telemetry framework, providing self-adaption capabilities based on data-driven mechanisms.
Collaborators	Provides telemetry and inventory information about the existing infrastructure (edge, cloud, HPC) and the deployed services to: - AI-enhanced Service Orchestrator (WP5T1AISO) - Resource Orchestrator (WP5T5RO) - Service Assurance (WP5T5SA)

	It also interacts with the various Enhanced Telemetry Agents (WP5T3ETA), collecting information and performing automatic reconfigurations for the telemetry infrastructure.
UCs	It will provide the required information for the operation of the AI-enhanced Service Orchestrator and Resource Orchestrator. Hence, it will contribute to UC applications' cognitive orchestration and deployment in the SERRANO platform.

Component ID	WP5T3AE
Name	Analytic Engine
High level description	It provides the analytic logic in the telemetry framework, integrating the design ML algorithms that will be able to: (i) characterize the performance of the infrastructure, (ii) estimate these metrics for different parts of the infrastructure by correlating telemetry measurements, and (iii) detect performance degradations and failures.
Collaborators	- Data Collector - Telemetry Controller - Notification Engine
UCs	It will analyse and correlate the monitoring information to provide orchestration and service assurance mechanisms an updated view of the infrastructure and service status and notifications for pro-active reconfigurations.

Component ID	WP5T3DC
Name	Data Collector
High level description	It collects the telemetry data from the available Enhanced Telemetry Agents and the various Monitoring Probes.
Collaborators	- Monitoring Probes - Enhanced Telemetry Agent - Analytic Engine - Persistent Monitoring Data Storage
UCs	It will feed the operational databased and Analytic Engine with telemetry data.

Component ID	WP5T3ETA
Name	Enhanced Telemetry Agent
High level description	It coordinates the operation of a specific set of Monitoring Probes that are responsible for monitoring the resources and the deployed applications. It is able to adjust their operation based on (i) the configuration commands from the Central Telemetry Handler and (ii) feedback from its Analytic Engine.
Collaborators	- Monitoring Probes - Central Telemetry Handler - Local Service Assurance
UCs	It will collect and forward to Central Telemetry Handler and Local Service Assurance mechanisms details about the characteristics and current status of available resources and deployed services under its administration domain.

Component ID	WP5T3LC
Name	Local Controller
High level description	It coordinates the operation of the Monitoring Probes according to the feedback by the internal Analytic Engine and the commands from the Central Telemetry Handler.
Collaborators	- Central Telemetry Handler - Analytic Engine - Monitoring Probes
UCs	It will orchestrate the operation of the monitoring probes that are under its administration. It will provide finer control over them compared to the central Telemetry Controller.

Component ID	WP5T3NE
Name	Notification Engine
High level description	It facilitates the exchange of events among the components of the hierarchical telemetry infrastructure. It also enables external services (primarily the Local Service Assurance mechanisms) to register for various performance-related events and automatically notify them accordingly.
Collaborators	- Analytic Engine - Local Service Assurance
UCs	It will enable self-optimization procedures, both reactively and proactively, at the local level without intervention from the central level mechanisms.

Component ID	WP5T3MLP
Name	Monitoring Probe
High level description	This component is responsible for the actual monitoring of resources and deployed applications. It will provide the necessary telemetry data to the telemetry framework.
Collaborators	- Enhanced Telemetry Agent
UCs	It will collect and forward to the other telemetry components information about the characteristics and current status of available resources and deployed services.

4.5 Data Broker

The SERRANO platform integrates the developed technologies and mechanisms, coupling them with orchestration and telemetry functionalities. It is a distributed and event-driven software platform that requires appropriate communication mechanisms to interconnect the individual components while ensuring scalability and loose coupling. Moreover, there are cases where a streaming platform is required to enable applications and internal components to handle and process efficiently real-time data streaming.

To this end, SERRANO incorporates the Data Broker service in the overall platform to support both message brokering and distributed streaming capabilities through the abstraction and integration of the appropriate message infrastructure. This service consists of two main components the Message Broker and Stream Handler. Following the Message-Oriented Middleware (MOM) architecture, the Message Broker provides the required mechanisms to enable the asynchronous communication and data transfer between the SERRANO platform components. The synchronous communication between the platform components is based on the APIs exposed by the components (i.e., following the Remote Procedure Call (RPC) pattern); hence there are no intermediate components.

In addition, the Stream Handler provides a distributed streaming service within the SERRANO platform that will allow publishing and subscribing to streams of records. This part of the messaging infrastructure can support high throughput and high-velocity data streams through a scalable, fault-tolerant communication-efficient framework. Hence, it enables the asynchronous communication between SERRANO platform components as well as deployed applications

The Message Broker component of the Data Broker is based on RabbitMQ [18], a well-known software platform that supports the required operations and provides increased scalability and security. The Stream Handler component of the Data Broker is based on Apache Kafka [17], an open-source distributed event streaming platform that is used high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. Figure 23 shows the key building blocks and their interactions with other SERRANO components, while Table 16 provides a high-level description for each component. A more detailed description of the Data Broker service is available in deliverable D5.3 (M15).

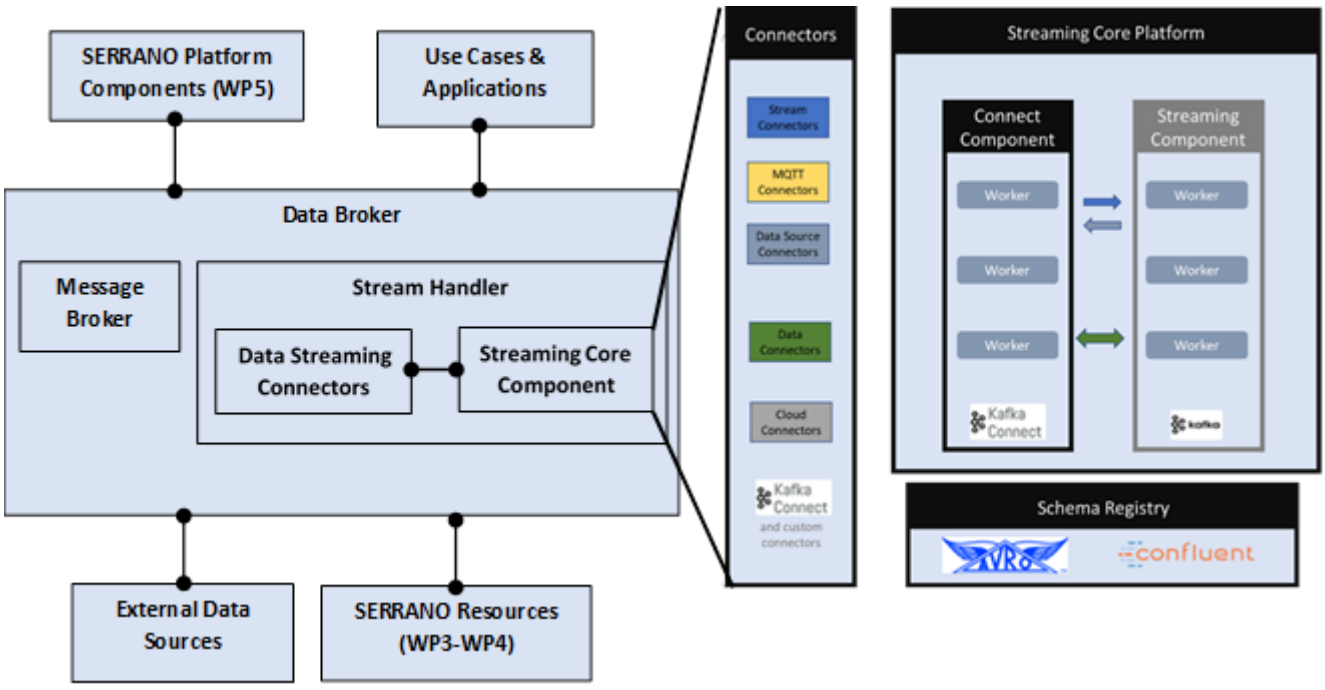


Figure 23: Data Broker core components and possible integrations with data sources and other infrastructure

Table 16: Description of data broker core components

Component ID	WP5T5MB
Name	Message Broker
High level description	It facilitates asynchronous communication between the various components in the SERRANO platform and the deployed applications. It abstracts the interaction with the actual message brokering middleware (i.e., RabbitMQ). The component leverages the publish and subscribe communication pattern to serve multiple senders and receivers simultaneously. It also supports message validation, transformation, and various message routing patterns.
Collaborators	- SERRANO software resources - SERRANO platform components - Deployed applications
UCs	It will handle the asynchronous inter-component communication towards a loosely coupled and scalable SERRANO platform. Moreover, it will facilitate the data transfer between the deployed applications and the SERRANO involved components.

Component ID	WP5T5SH
Name	Stream Handler
High level description	It facilitates high throughput communication between SERRANO components that participate in a publish-subscribe scheme to efficiently transfer information from publishers to multiple subscribers in the form of messages or data streams. Also, it connects various external data sources that provide stream data and makes them available to the SERRANO platform.
Collaborators	- External data sources - SERRANO platform components - Deployed applications
UCs	It handles messages and streams of data that require high throughput, special handling or transformation while moving between the components of the SERRANO Platform.

Component ID	WP5T5DSC
Name	Data Streaming Connectors
High level description	These support connectivity between Stream Handler and various data sources or communication protocols.
Collaborators	- Stream Handler - Streaming Core Component
UCs	Data sources that need to convert stored data into data streams. These can also produce Change Data Capture (CDC) streams for specific types of data sources.

Component ID	WP5T5SCC
Name	Streaming Core Component
High level description	It provides the ability to handle and process streams.
Collaborators	- Stream Handler - Data Streaming Connectors
UCs	This can support declarative or procedural definitions of handling streams. Also, different modes of operation are supported through its configuration.

5 SERRANO Architecture

5.1 SERRANO Overall Architecture

The vision of SERRANO will be implemented through a layered architecture, as depicted in Figure 24. In the selected design, each layer comprises a set of discrete components that interact horizontally and vertically to create the SERRANO platform.

The **Resource Layer** consists of heterogeneous edge, cloud, and HPC computational and storage resources encompassing the SERRANO hardware (Section 3.1) and software (Section 3.2) enhancements. That exceptional unification of highly diverse resources allows the SERRANO platform to cater to application and user constraints while calibrating the configuration of available resources. Furthermore, SERRANO will leverage the wealth of network-level monitoring information made available by SmartNICs. The key point is that network monitoring functions come for free since SmartNICs will be already deployed at key infrastructure nodes. This additional information will enable the resource orchestration mechanisms to take network-aware decisions, leading to a more efficient allocation of resources.

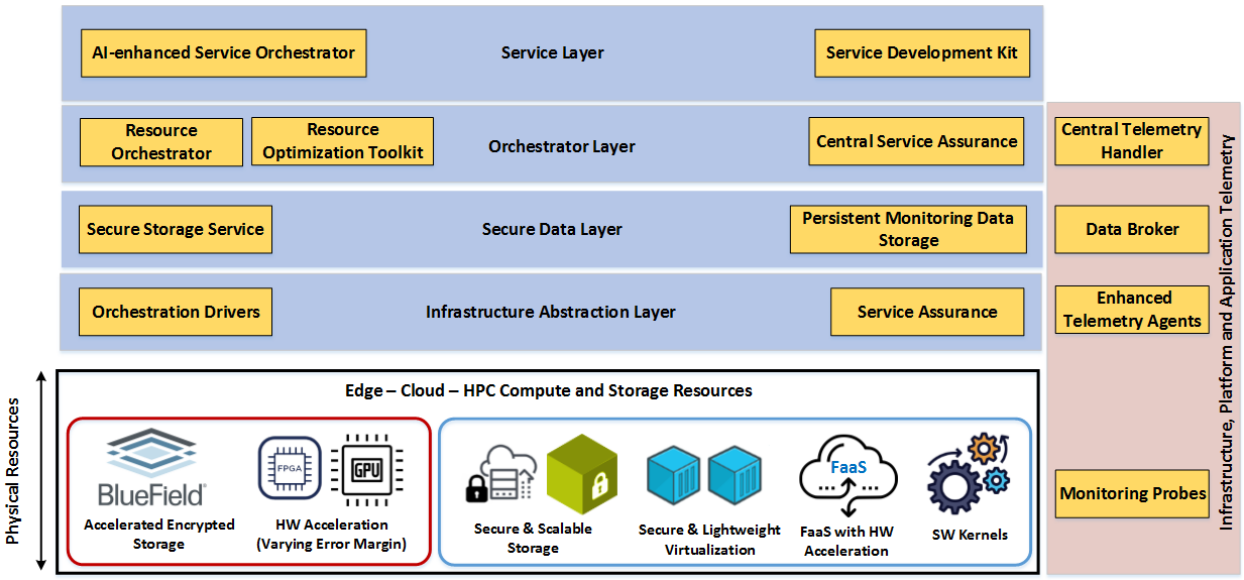


Figure 24: SERRANO high-level architecture

Across the SERRANO ecosystem resides the **Infrastructure, Platform and Application Telemetry stack** that collects metrics across the infrastructure and deployed applications. It is fuelling the platform and applications with data from all levels, enabling the implementation of data-driven and cognitive automation mechanisms. The hierarchical SERRANO network and cloud telemetry mechanisms implement these functionalities. The main components are the Central *Telemetry Handler*, the *Enhanced Telemetry Agents*, and *Monitoring Probes*. The *Central Telemetry Handler,* the root in SERRANO's telemetry hierarchy, collects through the *Enhanced Telemetry Agents* and *Monitoring Probes* the monitoring information to provide the

other components with valuable information and insights for the current state of the infrastructure and deployed applications. In addition, the *Data Broker* provides the required asynchronous inter-component communication within the SERRANO platform and connects external data sources, making them available to internal services. The component facilitates the loose coupling of SERRANO platform services and abstracts the management of applications' input and output data through the provision of asynchronous data transfer and distributed data streaming functionalities. Furthermore, it exposes the appropriate interfaces to enable both asynchronous communication functionalities (based on publish/subscribe pattern) and efficient distributed streaming capabilities.

The task of resource exposure to the upper layers is assigned to the **Infrastructure Abstraction Layer**, which abstracts the peculiarities of the management and interaction with the individual resources. This layer also provides a modular design to the whole SERRANO platform that ensures its extension through the immediate integration of new hardware and software platforms at the Resource Layer. The integration with the low-level resources at the distributed edge, cloud, and HPC infrastructures is achieved via the appropriate *Orchestration Drivers* that will provide the required mechanisms and interfaces to enable efficient and transparent deployment of services across the heterogeneous infrastructure. Furthermore, the *Service Assurance* at that level includes the various data-driven mechanisms, ranging from hardware-aware tuning techniques to application-specific optimizations (Section 4.3), that facilitate the identification of critical situations and activating self-driven adaptations for the deployed applications in each part of the overall SERRANO infrastructure.

The **Secure Data Layer** contains all the mechanisms for organizing all data resources within SERRANO and providing secure and easy-to-access interfaces for accessing and moving data across the individual platforms. The *Secure Storage Service* abstracts the required actions for edge and cloud storage resources, operating as a security access broker that guarantees and enforces privacy and security requirements on data. Finally, the *Persistent Monitoring Data Storage* allows the management of the historical monitoring data, necessary by the service assurance and remediation system.

The **Orchestration Layer** ensures efficient service orchestration and resource management in the disaggregated and heterogeneous SERRANO infrastructure. Taking as input from the Service Layer the application's high-level requirements and a candidate set of appropriate resource configurations, the SERRANO *Resource Orchestrator* allocates the proper configuration of hardware and data resources fulfilling the constraints of individual tasks. At the same time, it also automatically coordinates the necessary supplemental actions (e.g., transfer of required data). The *Resource Optimization Toolkit* provides network-aware joint computational and storage resource allocation and service placement algorithms, leveraging optimization and AI/MI techniques, emphasizing energy consumption, robustness, and latency. The *Central Service Assurance* manages the runtime lifecycle of each application deployment across the SERRANO heterogeneous infrastructure. According to service-specific needs, the infrastructure's current state, notifications from the *Central Telemetry Handler*, and the service assurance and remediation mechanisms at the infrastructure level, it can

automatically trigger proactively and reactively re-optimization actions to maintain the required performance level.

The **Service Layer** contains the *AI-enhanced Service Orchestrator* that automatically analyses applications to determine the possible deployment scenarios and translates the given application requirements (high-level requirements) into lower-level operational constraints and orchestration objectives. Moreover, the *SERRANO SDK* facilitates the development and deployment of innovative applications that fully leverage the provided functionalities. The SDK will include a set of well-defined APIs based on a transparent approach, where there are no hidden internal APIs.

A more detailed and elaborate diagram for the overall architecture of the SERRANO platform after the first iteration of the implementation and integration activities (M01-M18) is presented in Figure 25. It includes all the components that will be developed by the project technical work packages (WP3-6). In addition, the diagram depicts all the interactions and required information exchanges.

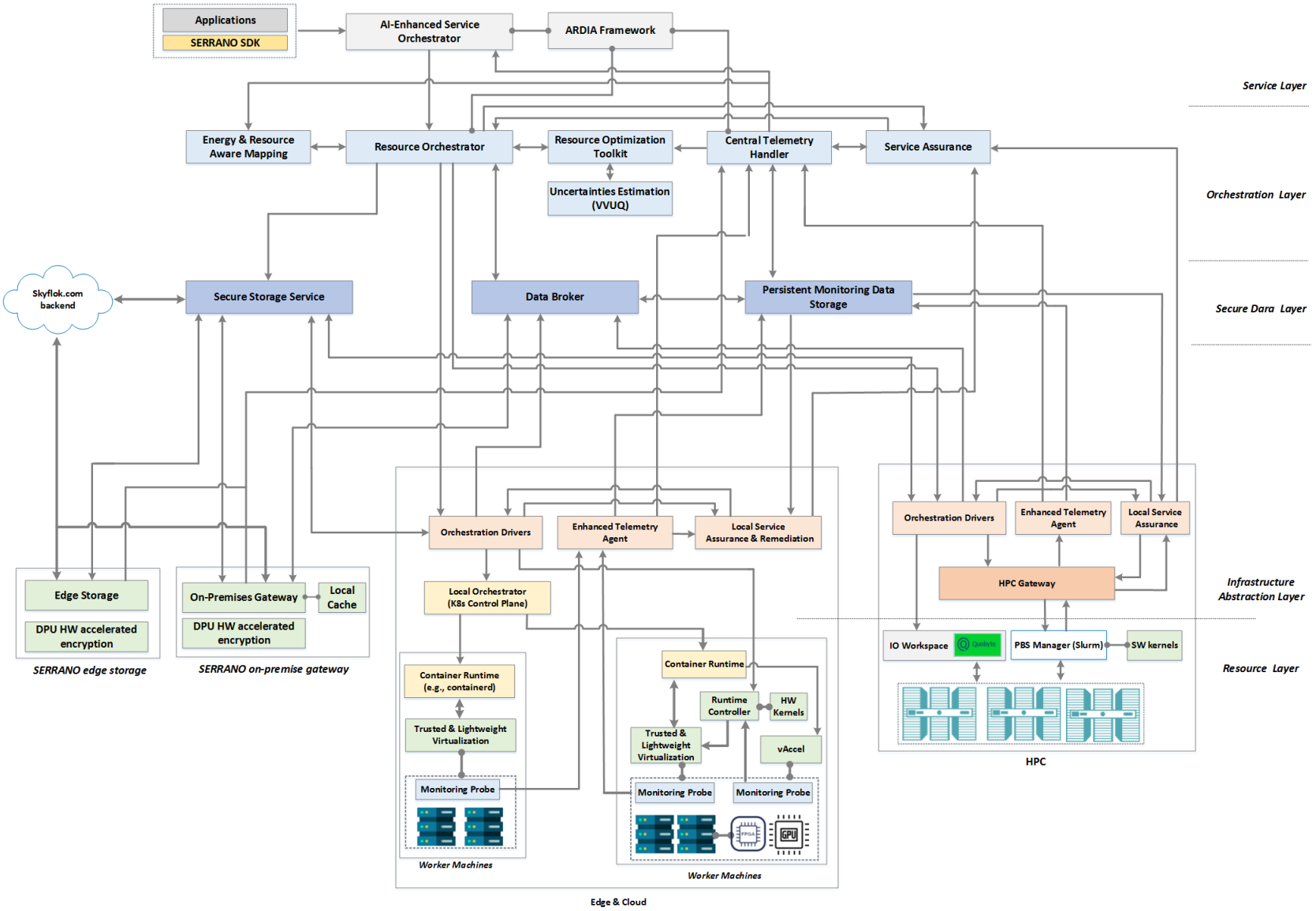


Figure 25: SERRANO detailed architecture

5.2 Information Flow View

This section complements the description of the SERRANO architecture by providing a set of information flow views that highlight the interactions and exchange of information between the core components and services of the SERRANO platform. These views focus both on the component-to-component interactions and application-to-system ones. In order to highlight the main functionalities of the SERRANO platform, the interactions are organized into three main control flow phases that correspond to the adopted application lifecycle:

- Application description and high-level requirements translation (steps a and b)
- Cognitive resource orchestration and transparent deployment (step c)
- Service assurance and dynamic adjustments (step d)

5.2.1 Application description and high-level requirements translation

The initial phase starts with the preparatory actions for the execution of the cloud-native applications in the SERRANO platform that involves: (i) proper annotation of application and (ii) definition of high-level infrastructure agnostic requirements (e.g., placement, security level, acceleration, etc.), and (iii) provision of the application-specific deployment descriptor(s) (e.g., container image, ports used, etc.).

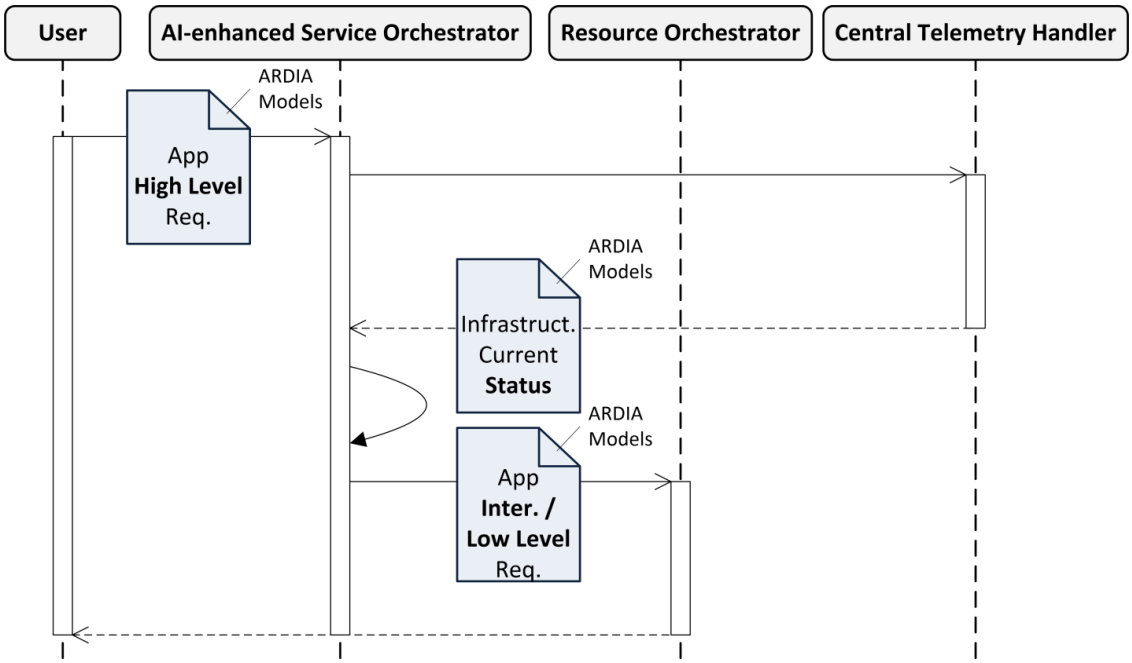


Figure 26: Interaction of AI-enhanced Service Orchestrator with the other components of the SERRANO platform

The formal description of high-level application requirements using the terminology specified in the ARDIA Framework (i.e., Application Model - Section 4.1.1) is necessary to identify the appropriate deployment scenario, as well as the constraints that the resources should satisfy, while considering the overall goals of the end user. Therefore, the AI-enhanced Service Orchestrator communicates with the Central Telemetry Handler, receiving telemetry data

regarding the status of the SERRANO infrastructure (expressed using the Telemetry Data Model), to revise and update the suggested resource constraints (intermediate or low-level requirements). These constraints are expressed based on the terminology specified in the ARDIA Framework (i.e., Resource Model) and are provided to the Resource Orchestrator, guiding its decisions.

Since the application's high-level requirements may be captured by more than one possible resource configuration, the AI-enhanced Service Orchestrator should provide potential Deployment Scenarios that the Resource Orchestrator could follow. Also, for each Deployment Scenario, the parameters/constraints included should be expressed using the elements of the Resource Model to facilitate the Resource Orchestrator decisions taking also into account the particular deployment descriptor(s) provided.

5.2.2 Cognitive resource orchestration and transparent deployment

The next phase includes two primary operations, the high-level cognitive resource orchestration and the launch of the transparent deployment procedures. In this phase, the orchestration mechanisms decide the placement of the application's workload in the individual edge, cloud, and HPC platforms that are unified under the control of the SERRANO platform. The orchestration mechanisms consider various criteria such as latency constraints, performance objectives, energy consumption, and applications' security requirements to allocate the application's workload to available platforms. They also form the deployment scenarios and formulate the appropriate infrastructure-specific deployment objectives for the distributed Orchestration Drivers.

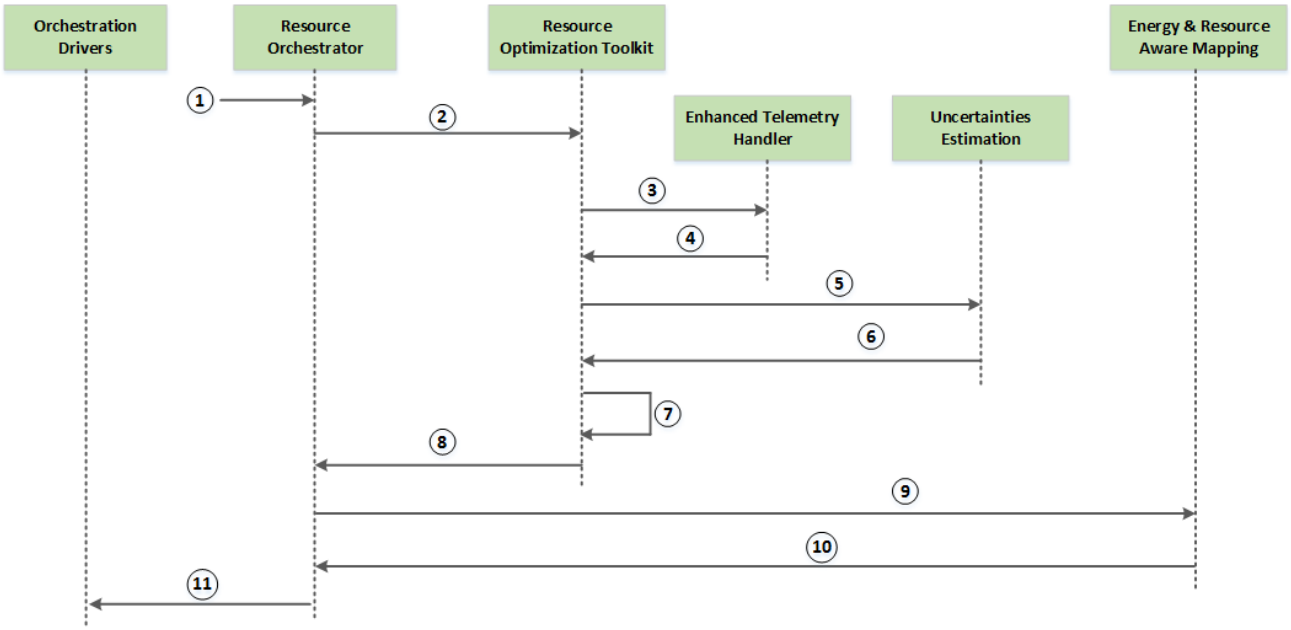


Figure 27: Cognitive resource orchestration operation within the SERRANO platform

Figure 27 highlights the involved components during the initial cognitive resource orchestration and their interactions:

- The SERRANO Resource Orchestrator receives the application description and a set of infrastructure-specific deployment requirements and constraints as input by the AI-enhanced Service Orchestrator.
- The Resource Orchestrator requests from the Resource Optimization Toolkit (ROT) to provide the most appropriate assignment of the application's workload to the individual platforms.
- The ROT retrieves from the Central Telemetry Handler updated information about the available resources in all federated edge, cloud, and HPC platforms.
- Based on the selected optimization algorithm, ROT decides on an initial workload placement that both satisfies user requirements and improves resource utilization.
- The Uncertainties Estimation provides through the VVUQ framework valuable insights and trade-offs regarding the uncertainties of the candidate hardware and software approximations for specific computationally intensive operations.
- The Energy and Resource Aware component estimates the impact of the ROT's decisions before the actual deployment in the infrastructure.
- The Resource Orchestrator prepares the specific deployment instructions and triggers the respective procedures.

When the placement decisions are delivered, the SERRANO platform coordinates the necessary actions for the actual deployment of the application, interacting with edge, cloud, and HPC resources (Figure 28). This phase includes the following actions:

- The Secure Storage Service creates the appropriate storage policies based on the instructions from the Resource Orchestrator.
- The Resource Orchestrator coordinates the movement of required data to the selected locations.
- The Central Telemetry Handler and the Central Service Assurance are notified to configure their mechanisms for following up on the deployment status of applications over the selected platforms.
- The Orchestration Drivers at the selected platforms receive the deployment instructions. Then, by interacting with the local orchestrator and the SERRANO-enhanced resources, they trigger the actual deployment of application's workloads with the selected runtime configurations.

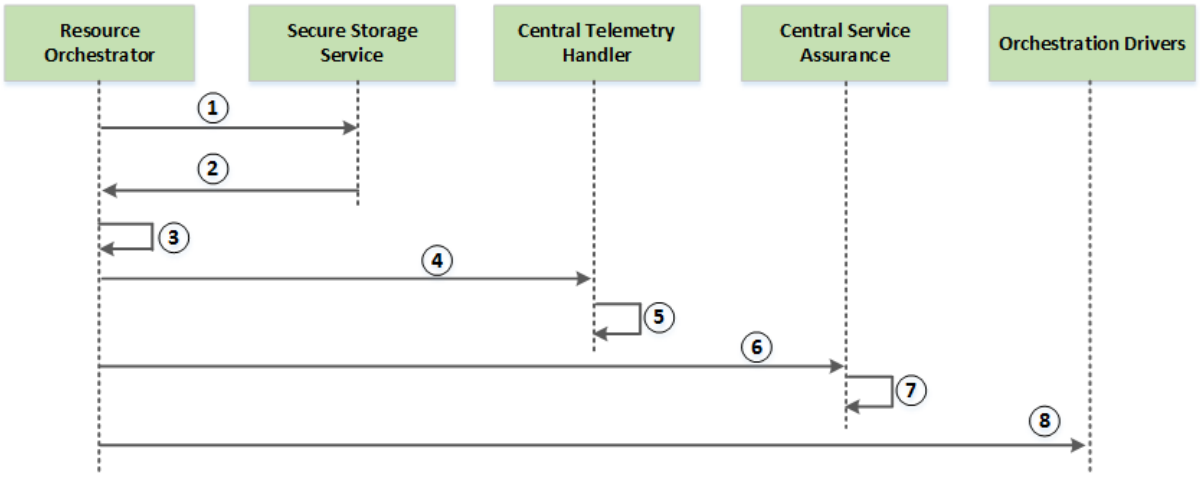


Figure 28: Transparent application deployment operation within the SERRANO platform

In the following subsections, we provide additional details for integrating SERRANO-enhanced resources running on edge, cloud, and HPC platforms.

5.2.2.1 HPC Integration

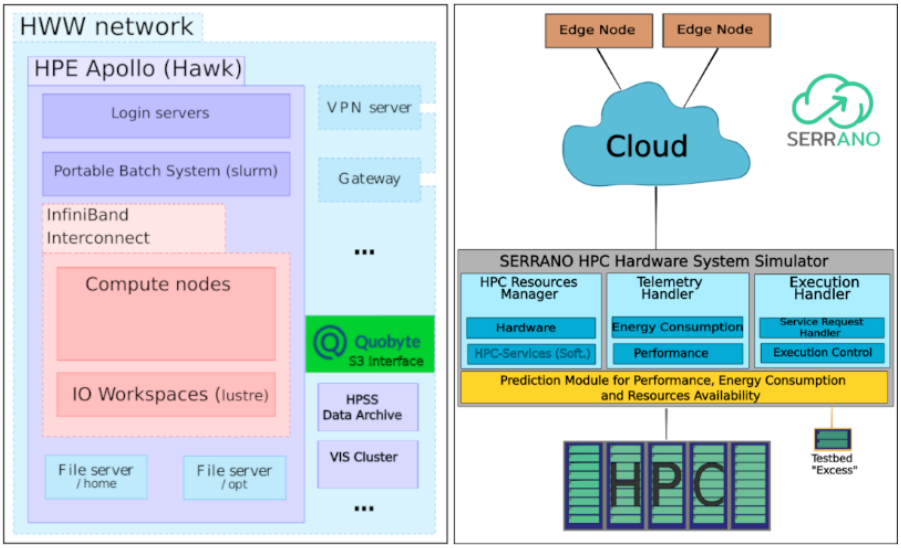
The Hawk supercomputer in HLRS is a desired platform for integrating HPC infrastructures in the context of SERRANO. More specifically, the models for the power and performance estimation will be calibrated for Hawk and the "EXCESS" test bed, which has similar hardware (CPU, RAM) and includes an extended set of telemetry sensors (for more details, see D2.1 "State of the Art Analysis Report").

Figure 29.a shows the internal HLRS network, so called HWW, that connects all HPC systems and their infrastructure at HLRS (for more details, see D2.2 "SERRANO use cases, platform requirements and KPIs analysis"). Each HPC system is a unique "device" with a unique isolated environment, while an HPC platform does not expose all the telemetry data. To this end, we developed an additional layer that serves as a gateway between the SERRANO platform and the HPC systems, namely HPC Gateway (in D2.3, we referred to it as HPC System Hardware Interface). This approach enables the immediate integration of HLRS resources within the SERRANO platform and makes the integration more universal and adaptable to various other HPC systems.

The HPC Gateway component (Figure 29.b) will provide:

- Description of the hardware (incl. IO resources);
- Description of the available HPC-Services, so-called SW kernels, e.g., solving of linear systems, applying signal processing filers on data;
- Nearly optimal hardware and software configuration for the HPC-Services (e.g., the optimal number of compute nodes, energy-aware CPU frequency);
- PBS scripts for the submission of a computational job (HPC service) into the HPC system queue;
- Fault tolerance handling;

- Available telemetry data or its estimation, if the HPC system does not provide it;



(a) (b)

Figure 29: (a) Internal HLRS's network HWW connects HPC resources at HLRS, (b) Main components of the HPC integration module within the SERRANO

Figure 30 depicts the interaction among the involved components for deploying an application in an HPC platform. The object file system "Quobyte" [19] will support the input and output data transfer between the HPC system and the SERRANO platform. Quobyte can facilitate the easy data exchange between HPC and non-HPC environments. The data is accessible through the standard Secure FTP (SFTP) interface and a native S3 storage interface. Although the bandwidth of the Quobyte interfaces is relatively high, it is several orders of magnitude smaller than that of the parallel file system of an HPC platform. This is currently an additional requirement for applications selected by SERRANO Resource Orchestrator for acceleration by HPC systems. However, this problem has been recognized and will be improved in HLRS and other HPC computing centres as far as technically possible in the future.

The Enhanced Telemetry Agent constantly monitors, through the HPC Gateway, the HPC platform and forwards the corresponding information to the Central Telemetry Handler. During the deployment, the Orchestration Driver (Section 4.2.1) receives the relevant instructions and forwards them to HPC Gateway to map them to HPC-specific commands. Moreover, if necessary, the Orchestration Driver moves the required input data to the HPC through the Secure Storage Service. Next, the deployment description is submitted to the Local Orchestrator (i.e., PBS Manager), while the Orchestration Driver instructs the Local Service Assurance to keep track of the submitted workloads through the HPC Gateway. Finally, the Orchestration Driver can send asynchronous messages to any other component in the SERRANO platform through the Data Broker (Section 4.5).

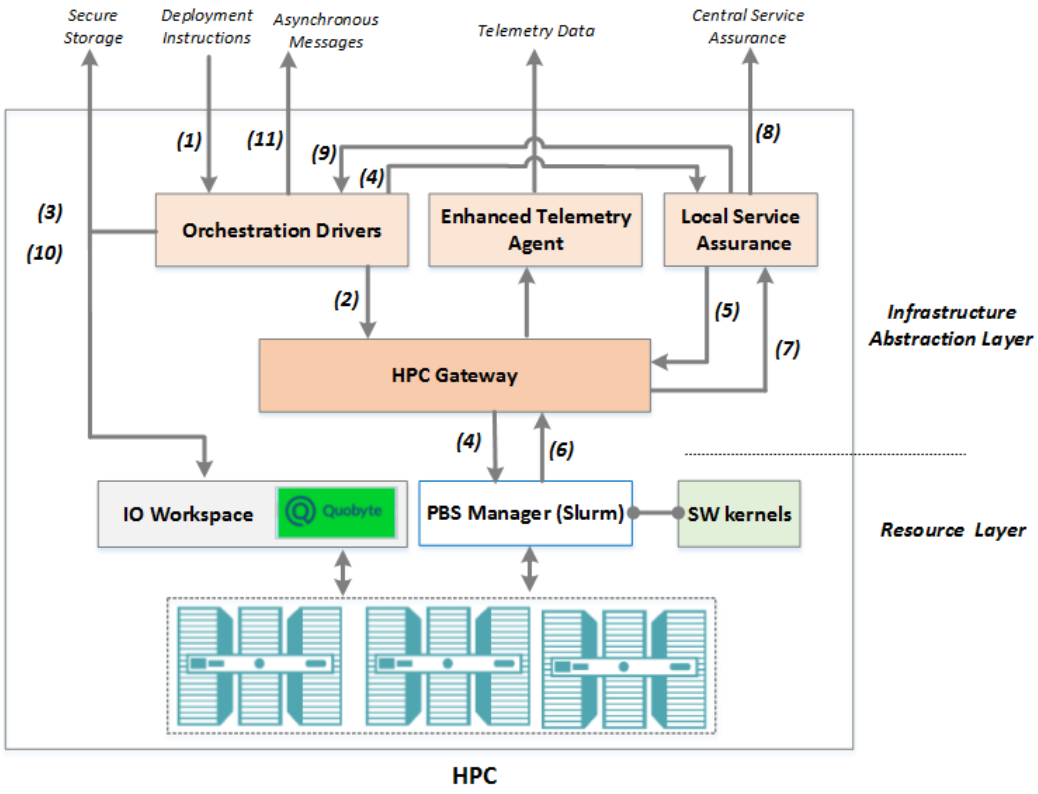


Figure 30: HPC integration and application deployment workflow

5.2.2.2 Edge and cloud integration

Figure 31 shows the interaction among the components that enable the transparent application deployment over edge and cloud platforms. In SERRANO, we consider that Local Orchestrators are based on existing and well-established solutions. More specifically, the orchestration platform for the edge and cloud platforms is the Kubernetes (K8s).

The Enhanced Telemetry Agents constantly monitor, through the available Monitoring Probes, the underline resources and available applications and forward the corresponding information to the Central Telemetry Handler. Among the developed probes, one can scrape metrics from the available GPUs (i.e., NVIDIA T4) and FPGAs (i.e., Xilinx Alveo U50 and U200). The probe is based on the Prometheus Timeseries DB and can be accessed by performing PromQL [21] queries to an endpoint. To expose GPU metrics, SERRANO uses the Data Center GPU Manager (DCGM) Exporter developed by NVIDIA. As for the FPGA metrics, it developed the Custom FPGA Exporter (FCE) for scraping metrics from the Xilinx Alveo devices available in SERRANO's platform. The FCE is based on the xbutil tool provided by Xilinx. The xbutil metrics are exposed through a Flask application to the exporter, responsible for translating them to the proper Prometheus format. More information concerning the exposed metrics is available in deliverable D6.3 (M18).

The actual deployment starts with the Orchestration Driver that translates the deployment instructions by the Resource Orchestrator to infrastructure-specific deployment descriptions that are passed to the Local Orchestrator (i.e., Kubernetes). Then, according to the SERRANO architecture, the Local Orchestrators perform the final resource orchestration that satisfies both the high-level orchestration objectives and ensures optimal use of the platform resources. D5.3 provides more technical details for the design and initial implementation of resource orchestration within the SERRANO platform. Moreover, through the SERRANO Secure Storage Service, the Orchestration Driver transfers the required input data.

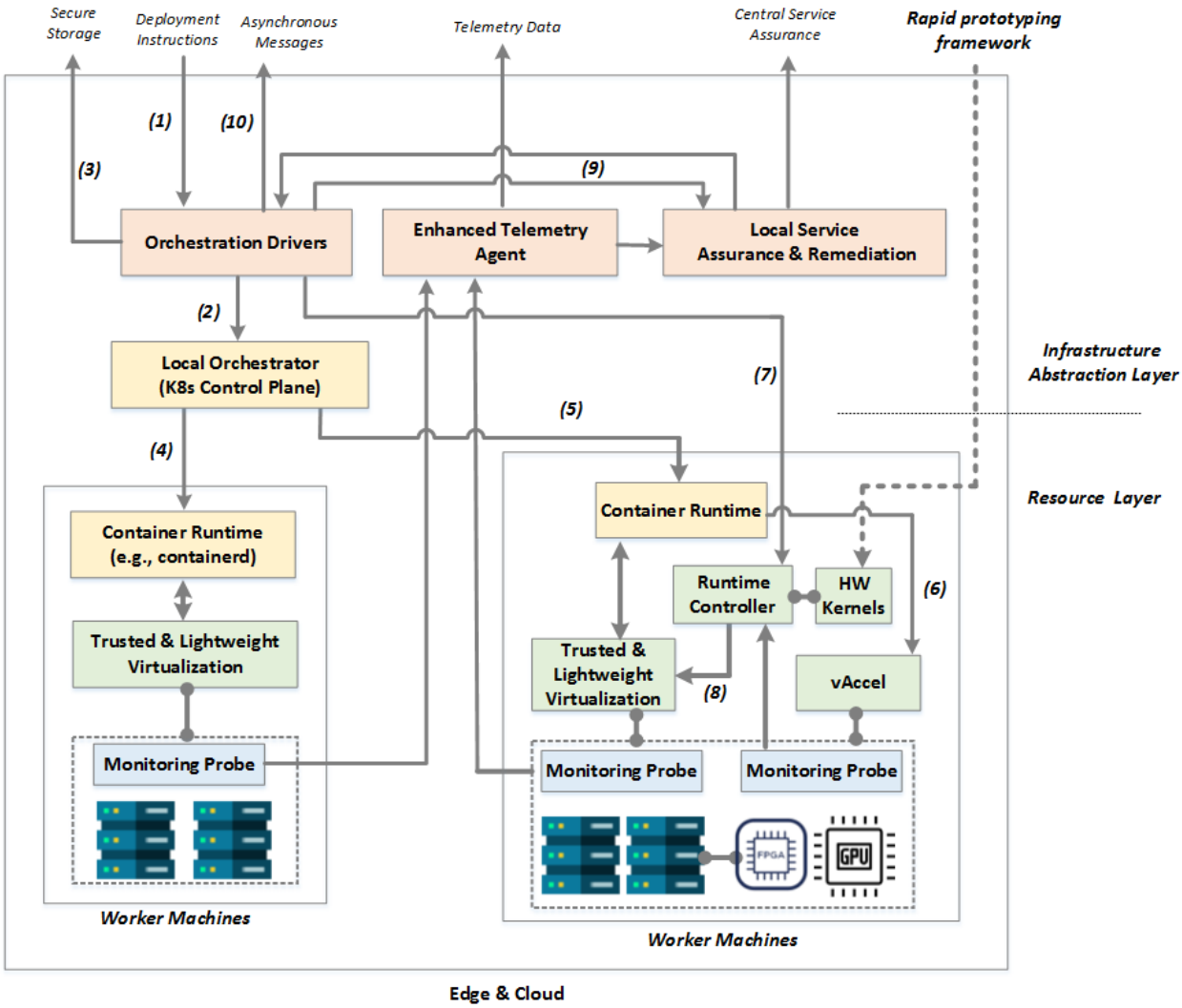


Figure 31: Edge and cloud integration and application deployment workflow

Since SERRANO aims to provide support for more than just containers (e.g., VMs, unikernels, etc.), it implements Trusted and Lightweight Virtualization mechanisms (Sections 3.2.2, 3.2.4) to provide the required glue layers that will enable the Local Orchestrator (i.e., Kubernetes) to deploy applications over edge and cloud nodes transparently and securely. For the secure and lightweight virtualization software resources, we also ensure compatibility with the upper and the lower parts of the systems software stack by being compatible with the Open Container Initiative (OCI) interface. On the upper-layer side, applications to be deployed are bundled as container images. On the node side, container runtimes are enhanced to support all modes of execution present in the SERRANO platform (containers, VMs, unikernels). In SERRANO, we also add the required functionality to the container runtime to support vAccel, the hardware acceleration framework introduced in Section 3.2.3, which enables access to hardware acceleration resources without the need for direct hardware/device access.

We group the above functionality under the control of the Local Orchestrator mechanisms that, through the SERRANO developments, provide: (a) the ability to spawn containers, VMs, and unikernels using hardware security extensions, (b) the ability to expose hardware acceleration functionality to workloads via the vAccel framework. Hardware accelerators are treated as separate components that expose acceleration functionality, which, in turn, are consumed as functions using a predefined interface.

Moreover, SERRANO targets an orchestration framework that manages the underlying hardware accelerators in an abstract and disaggregated manner. To this end, when the Orchestration Driver interacts with the Runtime Controller for approximate hardware kernels, it specifies the selected kernel and the requested quality-performance preferences (e.g., affordable error, energy). The approximate hardware kernels are created offline (i.e., depicted with dashed lines in the above figure) using a rapid prototyping framework (extended Plug&Chip) that aims to simplify the kernel designing process. Through this prototyping mechanism, the performance gains and the quality losses of all the designed approximate kernels are evaluated and the viable ones (in terms of satisfying QoS and accuracy constraints) constitute a *library of feasible approximate kernels.* Then, the Runtime Controller swaps between different application kernel versions (from the library of the *feasible approximate kernels*) based on the defined QoS constraints and/or interference/stressing scenarios. Each kernel version performs operations at a different level of approximation, trading off accuracy and performance against power.

The Service Assurance and Remediation mechanisms at the local level are instructed by the Orchestration Driver to keep track of the deployed applications and to ensure that the desired performance and quality requirements are met for each application. Finally, the Orchestration Driver can send asynchronous messages to any other component in the SERRANO platform through the Data Broker.

5.2.2.3 SERRANO Secure Storage Service

The SERRANO orchestration mechanisms interact with the SERRANO-enhanced Secure Storage Service to (i) create or update storage policies, (ii) perform storage-aware cognitive orchestration, and (iii) trigger the required data movement during the application deployment phase.

In the first case, the SERRANO Resource Orchestrator receives storage tasks (i.e., deployment descriptions that provide users' storage-related requirements) and decides how these tasks will be served. To this end, the orchestration mechanisms aim to match the requirements of these tasks with the most suitable storage locations, erasure coding configuration, and encryption scheme. This is done through the creation or selection of a storage policy. Deliverable D2.4 went into detail explaining this concept and provided some examples. In the second case, the orchestrator's application deployment decisions are guided by the availability of storage policies and resources. For example, information about the location of some data (after it has been stored using storage policies of SERRANO Secure Storage Service) may influence the central orchestrator's decisions regarding the assignment of computational workload to edge, cloud, and HPC resources.

The orchestration mechanisms use the provided interfaces (Storage Policy API, Telemetry API) for these two operations to interact with the Secure Storage Service. These interfaces provide details about the available storage resources (e.g., available data locations, the status of resources, cost, availability, latency, etc.) and enable the management of storage policies. The on-premises storage gateway is the component that exposes these APIs.

The last interaction enables the orchestration mechanisms to automatically perform the required data movement and is activated during the application deployment phase. For example, this requirement is essential for transparently integrating HPC platforms within the SERRANO ecosystem.

SERRANO applications access the SERRANO Secure Storage through the Secure Storage API (Section 6.2) when uploading and downloading files. The same interface also supports the orchestration services to automatically perform the required data movements during the application deployment phase. Figure 32 showcases the upload process, which involves the application uploading the file to the Gateway. The file is then cached, encrypted, and several (four in the example on the figure) erasure-coded fragments are created. Finally, these fragments are uploaded to a combination of cloud and edge locations (2 + 2 in the example). The Gateway performs the fragment uploads after authenticating with the Skyflok.com backend, which provides the upload links as pre-signed URLs, a technique described in Section 3.2.1.2.

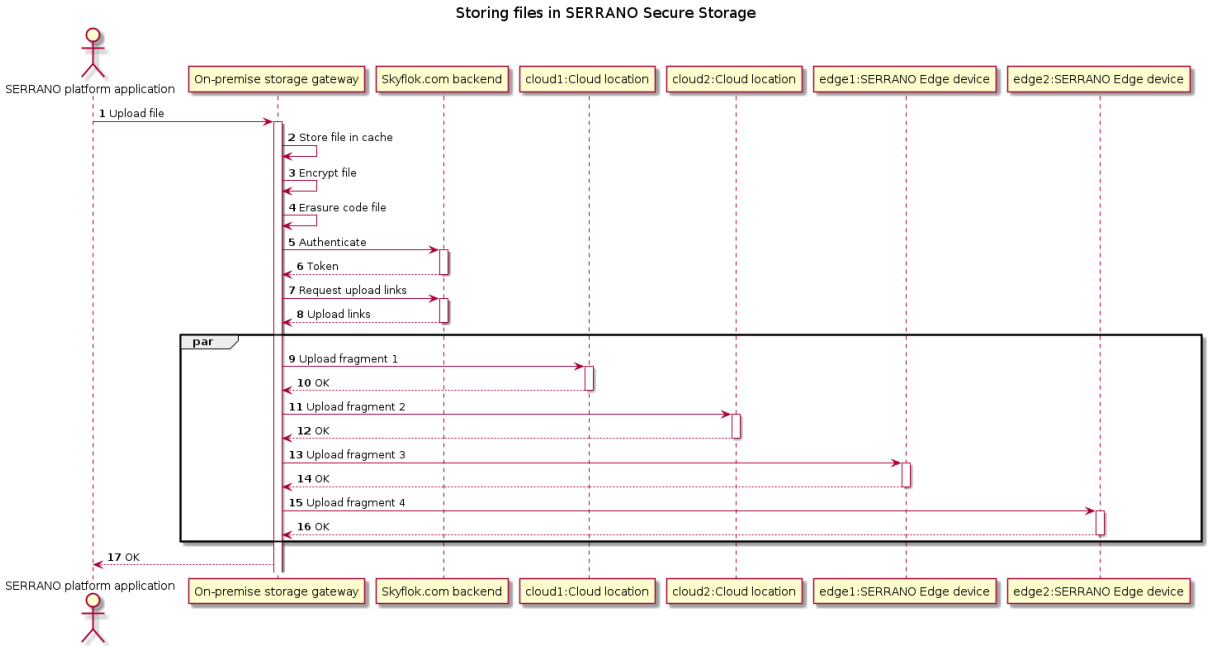


Figure 32: Sequence diagram showing file upload in SERRANO Secure Storage

Figure 33 showcases the download process. First, the SERRANO application requests the file using its filename, which acts as an identifier. Then, after authenticating with the Skyflok.com backend, the Gateway checks whether it has the file in its cache. If it does, it verifies with the backend that it has the latest version. If not, or if it is not in the cache, the Gateway requests pre-signed URLs to download the erasure-coded fragments. After sufficient fragments are retrieved from cloud and edge storage locations to decode, the file is decrypted and stored in the cache before being returned to the application.

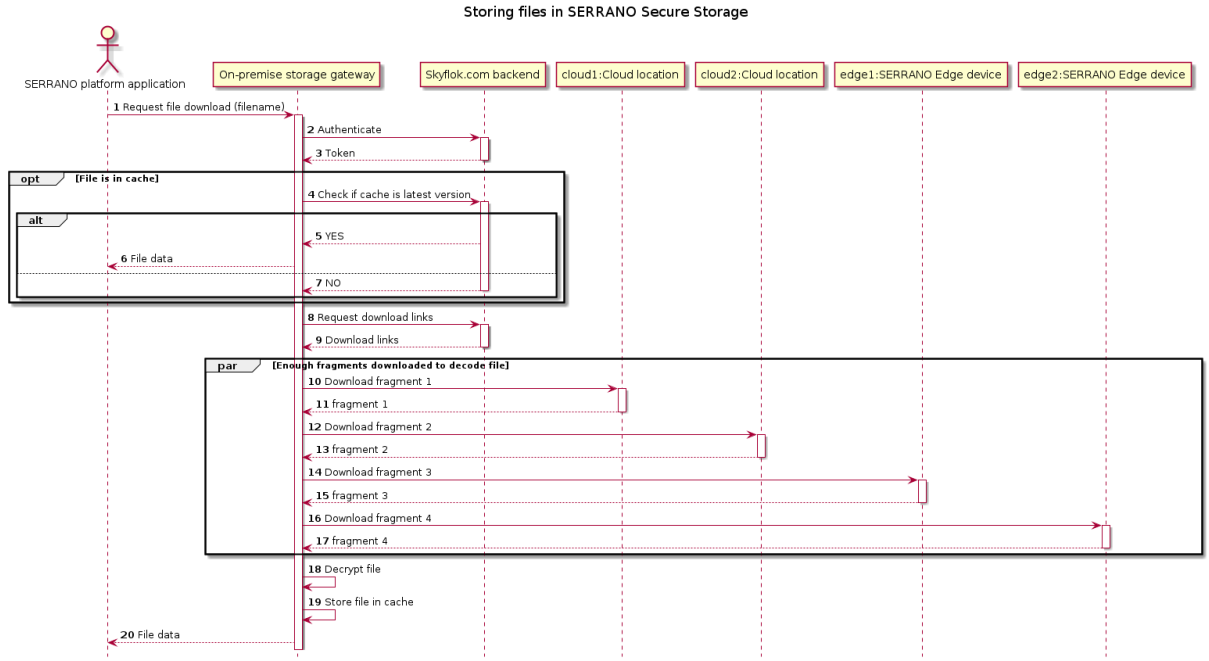


Figure 33: Sequence diagram showing file download from SERRANO Secure Storage

The actual movement of file data (both upload and download) is performed through the object storage interface (Section 6.2), which is a collection of interfaces implemented by the different Cloud Storage Services and the SERRANO edge devices.

5.2.3 Service assurance and dynamic adjustments

The final phase involves the service assurance flow that includes the SERRANO components responsible for safeguarding the deployed applications through the provision of dynamic and data-driven adjustments.

Figure 34 presents the active components, while this phase includes the following actions:

- The Service Assurance and Remediation (SAR) component gets by the Orchestration Drivers the description of the application execution plan that includes the committed resources and the service components to resource mappings.
- The SAR subscribes to Central Telemetry Handler to consume monitoring data that will be further pre-processed and ingested into the local data bus for internal processing (i.e., event detection for finding anomalies).
- The SAR, through the Event Detection Engine, constantly analyses the current state of the available resources and deployed applications.
- In case of anomaly detection, the SAR will issue and push alerts to the resource orchestration mechanisms through the Orchestration Driver to trigger a remediation action. If the detection methods allow a specific component anomaly identification, then a proper remediation plan can be predicted and provided to resource orchestration mechanisms.
- The Orchestration Driver contacts the local orchestrator that, based on the current state of the available resources and the remediation plan by the SAR, readjusts the initial deployment of the affected application.
- A special reconfiguration may occur if the application involves accelerated hardware kernels. Then, the Orchestration Driver also interacts with the respective Runtime Controller to specify desired execution parameters for the approximate hardware kernels.
- Finally, the SAR at the local level updates the service assurance mechanisms at the orchestration layer.

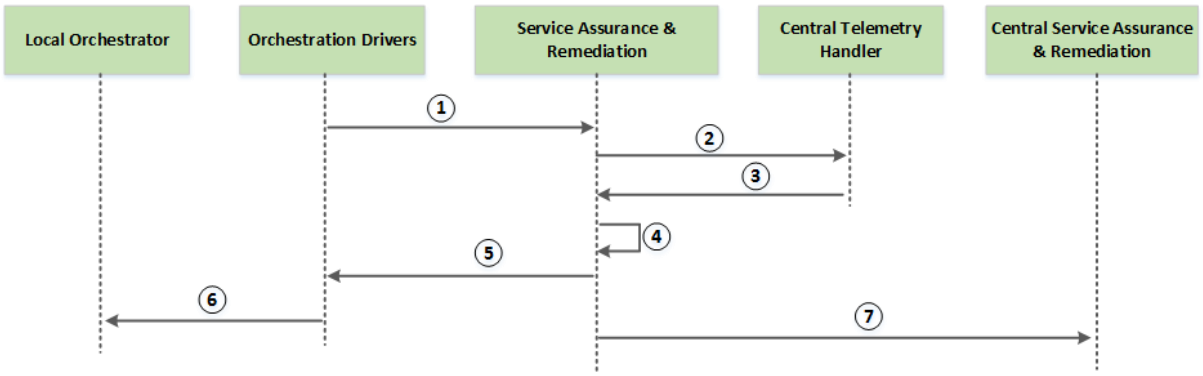


Figure 34: Service assurance and dynamic adjustments within the SERRANO platform

Additionally, the UCs that need to solve event detection problems can directly interact with the service assurance mechanisms. For example, the application will communicate with the Event Detection Engine (EDE) component through the SERRANO Message Broker or the Data Bus component of the SAR service and describe an event detection problem along with a specific scenario. Based on this description, the EDE will analyse specific monitoring data to detect anomalies and provide feedback to the application. Of course, this scenario is subject to UC specifications if the addressed problem can be solved using the Machine Learning techniques implemented by the EDE component.

6 Interfaces Specification

This section provides a high-level description of the interfaces required for the communication between the SERRANO components (Sections 3 and 4) developed in technical work packages (WP 3-5). The interfaces are defined jointly between the interface implementer and interface user while implemented in the context of the respective technical tasks. Moreover, their initial version has been utilized during the initial integration activities for the preparation of the initial release of the SERRANO platform, reported in deliverable D6.3 “The SERRANO integrated platform” (M18).

6.1 Service and Resource Orchestration Interfaces

In this subsection, we focus on the primary interfaces required for the communication between the core components at the service and orchestration layers. Specifically, the following tables provide a high-level description of the exposed interfaces by the AI-enhanced Service Orchestrator, Resource Orchestrator, Resource Optimization Toolkit, Uncertainties Estimation and Energy and Resource Aware Mapping. A more comprehensive and technical description for all the above interfaces is available at deliverables D5.1 “Abstraction models and intelligent service orchestration”, D5.2 “Algorithmic framework, performance and power models” and D5.3 “Resource orchestration, telemetry and lightweight virtualization mechanisms”.

Table 17: AI-enhanced Service Orchestrator Interface

Interface ID	WP5T1AISO-I
Description	This interface allows the translation of UC application profiles (including intents) to an intermediate level so that they can be effectively used by the Resource Orchestrator.
Component providing the interface	AI-enhanced Service Orchestrator
Consumer components	SERRANO platform user, Resource Orchestrator
Type of interface	REST
State	The AI-enhanced Service Orchestrator synchronously translates and enriches the UC applications requirements and goals (as provided by the SERRANO platform user) to the appropriate ones for every possible deployment scenario taking into account telemetry data collected so that the Resource Orchestrator can use them. This process has been described in the deliverable D5.1.
Constraints	Both input and output data should be expressed in JSON format based on the terminology specified in the ARDIA framework. The exact format of the messages is described in the deliverable D6.3.
Responsibilities	INNOV, ICCS

Table 18: Resource Orchestrator Interface

Interface ID	WP5T5RO-I
Description	This interface initiates the resource allocation and deployment of applications (e.g., as requested by the AI-enhanced service orchestrator). Moreover, it allows the service assurance mechanisms to trigger dynamic re-configurations of the already deployed applications.
Component providing the interface	Resource Orchestrator
Consumer components	AI-enhanced Service Orchestrator, Service Assurance
Type of interface	REST
State	Synchronous
Constraints	Both input and output data should be expressed in JSON format based on the terminology specified in the ARDIA framework.
Responsibilities	ICCS, INNOV, UVT

Table 19: Orchestration Drivers Interface

Interface ID	WP5T5OD-I
Description	It supports the implementation of hierarchical orchestration and application deployment within the SERRANO platform. It exposes the required methods to enable the interaction between the Resource Orchestrator and the local orchestration platforms. The Resource Orchestrator sends instructions for new applications deployment or reconfiguration for applications running on the SERRANO resources.
Component providing the interface	Orchestration Driver
Consumer components	Resource Orchestrator, Local Orchestrators at individual infrastructures
Type of interface	REST and AMQP
State	Synchronous and Asynchronous
Constraints	The description of deployment instructions should be generic and has to follow the ARDIA framework specifications.
Responsibilities	ICCS, NBFC

Table 20: Resource Optimization Toolkit Interface

Interface ID	WP5T2ROT-I
Description	The interface enables the Resource Orchestrator to interact with the Resource Optimization Toolkit to retrieve deployment decisions from the available multi-objective resource allocation algorithms.
Component providing the interface	Resource Optimization Toolkit

Consumer components	Resource Orchestrator
Type of interface	REST and AMQP
State	Synchronous and Asynchronous
Constraints	Both input and output data should be expressed in JSON format based on the terminology specified in the ARDIA framework.
Responsibilities	ICCS

Table 21: Uncertainties Estimation Interface

Interface ID	WP4T2HPC-I
Description	It will provide insights and trade-offs regarding the hardware and software approximations for particular computationally intensive operations.
Component providing the interface	HPC Framework VVUQ, SERRANO HPC Gateway
Consumer components	Orchestration services, internal components, UCs and other applications that need to consume the HPC services exposed by the HPC systems.
Type of interface	Binary protocols, C++ API
State	Asynchronous
Constraints	Applications should specify the error metric (e.g., prediction accuracy, the norm of the error, or any specific code) used to measure the accuracy/error. In addition, maximum error thresholds should be specified for applications.
Responsibilities	HLRS is responsible for the development. The estimation of the error and its effect on the results will be coordinated with the respective use case partner (IDEKO, InbestMe). ICCS, AUTH support the integration into the Orchestration services.

Table 22: Energy & Resource Aware Mapping Interface

Interface ID	WP5T4EMT-I
Description	SERRANO Energy & Resource Aware mapping concerns the efficient use of SERRANO platform components, particularly the HPC systems.
Component providing the interface	SERRANO HPC Gateway
Consumer components	Orchestration services, internal components, UCs and other applications that need to consume the HPC services exposed by the HPC systems, which are connected via the SERRANO HPC Gateway.
Type of interface	Binary protocols
State	Asynchronous
Constraints	Both input and output data should be expressed in JSON format based on the terminology specified in the ARDIA framework. Coupling to the underlying hardware for energy and power measurements.
Responsibilities	HLRS, ICCS, INNOV

6.2 Secure Storage API and Telemetry Interfaces

The Secure Storage API (Table 23) will enable both the applications deployed to the SERRANO platform and the platform services (Section 4) to interact with the on-premises storage gateway. It follows the de-facto standard set by Amazon with its S3 object storage service, a design choice that allows easy integration with existing applications designed to work with this API. It will support the most important operations of S3 but does not seek to provide complete coverage of the S3 feature set. Instead, it is an interface tailored to meet the requirements of the SERRANO platform and its applications, represented by the project's three use cases (UCs).

Table 23: Secure Storage API

Interface ID:	WP3T2DSS-I
Description	This interface provides a way to upload and download files to/from the SERRANO-enhanced Storage Service.
Component providing the interface	On-premises storage gateway
Consumer components	SERRANO applications as well as platform services
Type of interface	REST
State	Synchronous
Constraints	-
Responsibilities	Chocolate Cloud is responsible for the implementation of the interface

The on-premises storage gateway (Gateway) accesses the objects on the cloud storage locations through either proprietary interfaces, S3 or S3-compatible interfaces, or the Swift API. In addition, objects on SERRANO edge devices are also accessed through an S3-compatible interface, as provided by the MinIO S3 Gateway [20]. Table 24 summarizes the REST interfaces provided by each storage location.

Table 24: List of REST interfaces used to access the storage locations

Storage location	Interface
Amazon S3	S3
Google Cloud Storage	Proprietary API
Azure Blob Storage	Proprietary API
OVH	SWIFT API
Exoscale	S3-compatible
IONOS	S3-compatible
Dunkel	S3-compatible
Outscale	S3-compatible
SERRANO edge devices	S3-compatible

The SERRANO edge devices as well as the Skyflok.com backend (through the Gateway) must expose an interface that allows the SERRANO orchestration and telemetry services to retrieve information about the characteristics and current state of storage locations (Table 25).

Table 25: Storage location telemetry API

Interface ID:	WP3T2DSSSLT-I
Description	This interface provides information about the characteristics and current state of cloud and edge storage locations.
Component providing the interface	The On-premises storage gateway provides information about cloud locations, The SERRANO edge devices provide information about themselves.
Consumer components	Central Telemetry Hander, Resource Orchestrator.
Type of interface	REST
State	Synchronous
Constraints	Requests must be performed using HTTPS, HTTP is not supported.
Responsibilities	CC, ICCS

Next, we present the specifications for the provided interfaces by the main components of the SERRANO telemetry mechanisms and data broker services.

Table 26: Central Telemetry Handler Interface

Interface ID	WP5T3CTH-I
Description	This interface allows the main components of the SERRANO platform to receive monitoring data from the available edge, cloud, and HPC resources, along with the status of the deployed applications. The initial design also considered the exchange of notification events through that interface. However, the Stream Handler interface (Table 30) supports this functionality in the final design.
Component providing the interface	Central Telemetry Handler
Consumer components	AI-enhanced Service Orchestrator, Secure Storage, Resource Optimization Toolkit, Service Assurance
Type of interface	REST, AMQP
State	Synchronous/Asynchronous Depending on the monitoring data that the interacting components retrieve, the communication can be synchronous or asynchronous, hence the Central Telemetry Handler will support both paradigms.
Constraints	The description of available resources should follow the ARDIA framework specifications.
Responsibilities	ICCS, CC, HLRS, INNOV, UVT

Table 27: Enhanced Telemetry Agent Interface

Interface ID	WP5T3ETA-I
Description	This interface is responsible for forwarding the monitoring information across the hierarchical telemetry infrastructure. It also provides methods for managing the collection, pre-processing, and retrieval of telemetry data.
Component providing the interface	Enhanced Telemetry Agent
Consumer components	Central Telemetry Handler, Orchestration Drivers, Service Assurance and Remediation, Monitoring Probes
Type of interface	REST, AMQP
State	Synchronous/Asynchronous Depending on the information that the interacting components provide and retrieve the communication can be synchronous or asynchronous.
Constraints	The description of available resources should follow the ARDIA framework specifications.
Responsibilities	ICCS, NBFC, UVT

Table 28: Persistent Monitoring Data Storage Interface

Interface ID	WP5T5PMDS-I
Description	It will enable a storage service for the historical monitoring data using a distributed data store (i.e., a NoSQL technology with time-series data storing support). It will also expose a multi-purpose communication interface to interact with the data store (i.e., insert and query operations will be supported).
Component providing the interface	Service Assurance and Remediation
Consumer components	Internal components, UCs and other applications or components that need to consume the services exposed by the component.
Type of interface	REST, AMQP (optional), Binary protocols (optional)
State	Synchronous
Constraints	The format of the messages exchanged with the persistent monitoring data storage interface will be fixed and provided by the interface specifications.
Responsibilities	UVT, ICCS

Table 29: Message Broker Interface

Interface ID	WP5T5MB-I
Description	Message Broker Interface
Component providing the interface	The Message Broker supports communication using protocols specialized for message exchange (i.e., AMQP, MQTT).
Consumer components	SERRANO Platform components, use cases and applications, external data sources, SERRANO resources
Type of interface	AMQP, MQTT
State	Asynchronous

Constraints	Depending on the communication channel security, other components connecting to this interface might need an X.509 or similar certificate to ensure secure communication for sensitive data.
Responsibilities	INTRA, ICCS

Table 30: Streaming Core Interface

Interface ID	WP5T5SC-I
Description	Streaming Core Interface
Component providing the interface	The Streaming Core Interface supports communication with the Streaming Core Component via compatible client libraries in other components.
Consumer components	Data Streaming Connectors, SERRANO Platform components, use cases and applications, external data sources, SERRANO resources
Type of interface	TCP (Kafka wire protocol)
State	Stream
Constraints	Depending on the communication channel security, other components connecting to this interface might need an X.509 or similar certificate to ensure secure communication for sensitive data.
Responsibilities	INTRA

Table 31: Data Streaming Connector Interface

Interface ID	WP5T5DSC-I
Description	Data Streaming Connector Interfaces
Component providing the interface	The Data Streaming Connectors enables communication with the Streaming Core component using various protocols.
Consumer components	SERRANO Platform components, use cases and applications, external data sources, SERRANO resources.
Type of interface	MQTT, REST, any other type supported by a custom or open-source connector.
State	Asynchronous, Stream
Constraints	Depending on the communication channel security, other components connecting to these interfaces might need an X.509 or similar certificate to ensure secure communication for sensitive data.
Responsibilities	INTRA

6.3 Service Assurance and Resource Management Interfaces

The Service Assurance and Remediation system will expose the appropriate communication interface to facilitate external interactions with the SERRANO components. The interface will be flexible and support different protocols and communication models to facilitate easy integration with the other SERRANO platform components.

Table 32: Service Assurance Interface

Interface ID	WP5T5SAR-I
Description	SERRANO Service Assurance and Remediation Interface will expose internal functionalities to other platform components.
Component providing the interface	Service Assurance and Remediation
Consumer components	Orchestration services, internal components, UCs and other applications that need to consume the services exposed by the component.
Type of interface	REST, AMQP, Binary protocols
State	Synchronous/Asynchronous Based on the components that need to interact with the service assurance, the communication can be synchronous or asynchronous, the service assurance component supports both paradigms.
Constraints	The format of the messages exchanged with the service assurance component is predefined, but it can be extended and adapted based on the specifications provided by the components that needs to interact with.
Responsibilities	UVT, ICCS

The optimization and tuning techniques described in Sections 4.3.1 and 4.3.2 are implemented as extension plugins for the Plug&Chip rapid prototyping and DMM4FPGA frameworks and can be either used offline (users can download the tools and use them on their premises) or leverage the APIs that expose these functionalities as services through the SERRANO platform. Moreover, hardware accelerators (GPUs and FPGAs) are exposed as Kubernetes resources by leveraging the official device plugins provided by NVIDIA and Xilinx. The tables below present the exposed interfaces for utilizing the optimization services and hardware acceleration devices through the SERRANO platform.

Table 33: Plug&Chip Interface

Interface ID	WP4T3PC-I
Description	SERRANO rapid prototyping interface will expose the extended Plug&Chip framework's functionality to end-users.
Component providing the interface	Independent SERRANO tool
Consumer components	UCs and other applications that need to consume the resources exposed by the component.
Type of interface	Socket interface
State	Synchronous
Constraints	Users leveraging this API should provide a C/C++ source code along with their application's requirements
Responsibilities	AUTH

Table 34: DMM4FPGA Interface

Interface ID	WP4T3DF-I
Description	SERRANO's interface that will expose the DMM4FPGA framework's functionality to end-users.
Component providing the interface	Independent SERRANO tool
Consumer components	UCs and other applications that need to consume the resources exposed by the component.
Type of interface	Socket interface
State	Synchronous
Constraints	Users leveraging this API should provide a C/C++ source code
Responsibilities	AUTH

Finally, the following tables provide the specifications for the required interfaces that enable the integration of the SERRANO-enhanced software and hardware resources with the SERRANO orchestration mechanisms.

Table 35: Hardware Accelerators Interface

Interface ID	WP4T1HWRT-I
Description	SERRANO Hardware accelerators API will expose accelerator resources (e.g., GPUs, FPGAs) to the orchestrator and the end-users/applications. In addition, it will expose accelerators in case of end-users explicitly request GPU/FPGA resources. However, transparent HW acceleration capabilities will still be able through the vAccel framework.
Component providing the interface	Local orchestrator of HW acceleration resources
Consumer components	Global orchestrator, UCs and other applications that need to consume the resources exposed by the component.
Type of interface	REST, command-line interface
State	Synchronous
Constraints	The format of the messages exchanged should follow the structure defined by the local orchestration mechanisms.
Responsibilities	AUTH

Table 36: HPC Services Interface

Interface ID	WP4T2HPC-I
Description	SERRANO HPC Services interface provides the set of the HPC-hosted operations for the UCs (e.g., Kalmar and FFT Filters, solving systems of linear equations).
Component providing the interface	SERRANO HPC Gateway
Consumer components	Orchestration services, UCs and other applications that need to compute large problems.
Type of interface	Binary protocols, C/C++ API, PBS script
State	Synchronous/Asynchronous Based on the requirements of the UCs the executions of the HPC services can be synchronous or asynchronous.
Constraints	The format of the data exchanged with the HPC services is predefined but it can be extended and adapted based on the specifications provided by the components that needs to interact with.
Responsibilities	HLRS is responsible for the development of the related components and interfaces. In coordination with IDEKO and InbestMe, the services will be defined and will be adapted to the respective UC. ICCS and AUTH support the interaction with the orchestration services.

Table 37: Trusted and Lightweight Virtualization Interface

Interface ID	WP5T5TLV-I
Description	SERRANO Trusted & Lightweight Virtualization interface exposes workload execution functionality to the orchestrator and the end-users/applications. It reflects the type of workload to be executed (essentially a container image) as well as the needed resources.
Component providing the interface	Local orchestrator
Consumer components	Orchestration Drivers, Local Orchestrators, UCs and other applications that need to spawn tasks in the SERRANO platform resources.
Type of interface	REST, command-line interface
State	Asynchronous
Constraints	The format of the exchanged messages should follow the structure defined by the local orchestrator. To interact with the SERRANO orchestration mechanisms as well as other local orchestrators we choose to make this interface OCI-compatible.
Responsibilities	NBFC

6.4 SERRANO Service Development Kit

The success and wide acceptance of any platform largely depend on the functionality and services offered to end users and application developers. Moreover, it is widely recognized that a company's competitiveness depends directly on its capacity to realize new ideas with the minimum time to market. SERRANO will deliver a complete Service Development Kit (SDK) along with all the required functionality to effectively support the development and deployment of innovative applications that fully leverage the provided innovations. SERRANO SDK will include a set of well-defined APIs, adopting a transparent approach where there are no hidden internal APIs.

Table 38: SERRANO SDK

Interface ID	WP6T1SDK-I
Description	The SERRANO SDK exposes the developed APIs through a library.
Component providing the interface	SERRANO SDK
Consumer components	Components that use the SERRANO APIs will be able to do so through the SERRANO SDK
Type of interface	Python library
State	It supports the states that the SERRANO APIs can support.
Constraints	The constraints of SERRANO APIs also apply to SDK. Also, the Python SDK will be usable in a Python App as a requirement that can be installed using pip. The functionality provided by this interface depends on the provided functionality by the APIs.
Responsibilities	INTRA

7 SERRANO Platform Deployment View

To highlight the capabilities of the secure, disaggregated, and accelerated SERRANO platform in supporting highly-demanding, dynamic and safety-critical applications, the project will demonstrate three high impact use cases (UCs) across different domains with heterogeneous needs. These UCs include (i) secure cloud and edge storage over a diversity of resources, (ii) secure launching of a large number of FinTech operations, and (iii) advanced machine anomaly detection in manufacturing for Industry 4.0.

In what follows, we provide a high-level description of the separate deployments that SERRANO will set up for evaluating the UCs. D6.1 “Use cases technological developments” (M16) and D6.2 “KPIs and evaluation methodology” (M18) describe the initial technological developments (services, mechanisms, interfaces) built in each use case and provide a general set of guidelines for evaluating the SERRANO platform and UCs, respectively.

7.1 UC1: Secure Storage

The principal components of the Secure Storage use case (UC1) are shown on Figure 35.

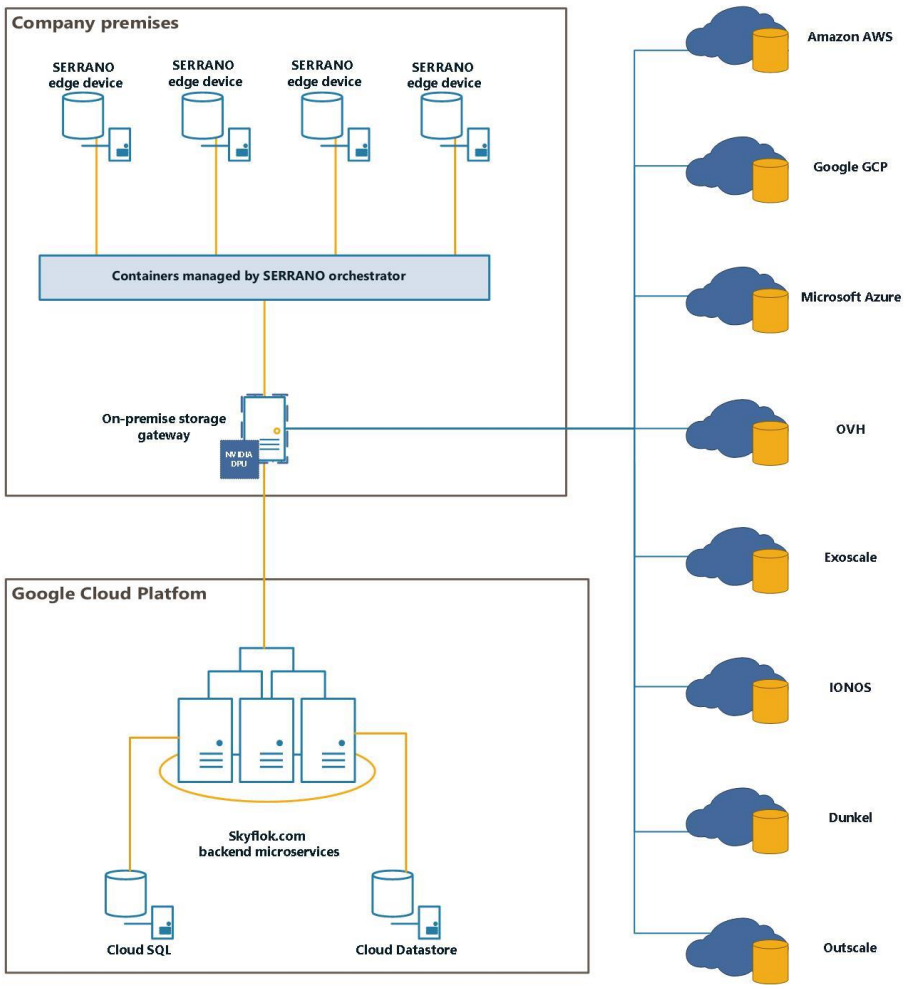


Figure 35: Secure Storage use case deployment overview

The on-premises storage gateway and the SERRANO edge devices will run as containerized applications on the respective devices, deployed on the customer's premises. These will be managed by the SERRANO platform.

The Skyflok.com backend, developed before the project, is deployed on the Google Cloud Platform (GCP) as a series of micro-services running in the App Engine standard environment. Persistence is achieved using a combination of Cloud SQL (GCP's SQL server) and Cloud Datastore (a NoSQL database). UC1 also enables data storage on a multitude of cloud locations, also shown on the figure. While SkyFlok supports several other smaller European cloud services as well, the selected ones bring good variety in terms of cost, performance and availability. Together, these provide a total of 57 distinct storage locations across several continents. The SERRANO platform will decide the combination of edge and cloud resources to serve the storage tasks.

7.2 UC2: Fintech Analysis

Figure 36 depicts the deployment of the FinTech use case (UC) that focuses on portfolio optimisation. The selected deployment includes both on-premises servers as well as cloud servers. The SERRANO orchestration mechanisms will handle the entire deployment of the UC-related services.

The on-premises services will mainly execute operations that are not containerized in the context of SERRANO but are required by the UC. They include obtaining market data, creating investment profiles, portfolio rebalancing, and order operations. In addition, multiple microservices instances will be deployed into nodes that support the SERRANO software and hardware enhancements. Hence, some microservices will use the available hardware accelerators on the deployed nodes. For example, market analysis, forecasting, and backtesting would benefit from the available acceleration on a particular SERRANO-enhanced node, while the others can be cognitively deployed in other nodes.

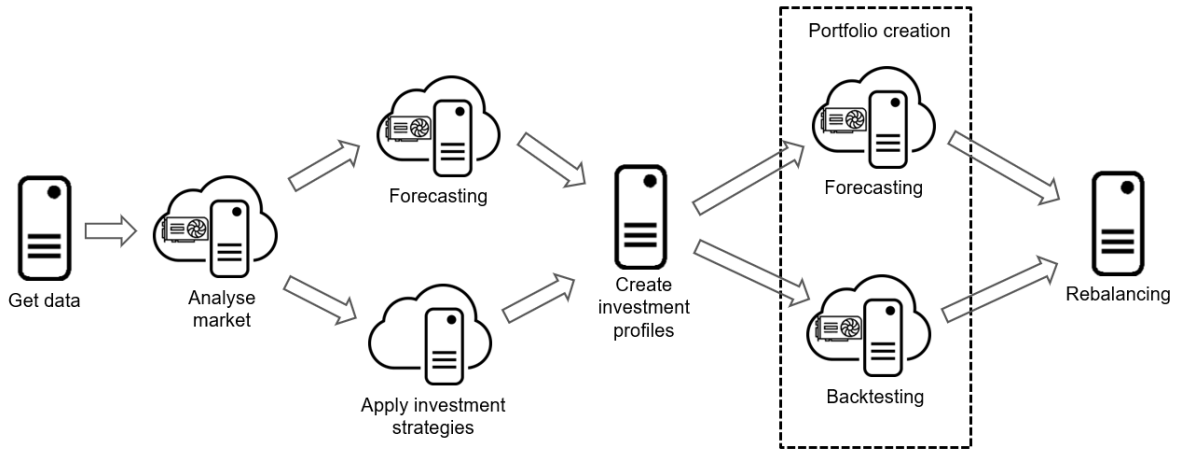


Figure 36: FinTech portfolio optimisation use case deployment overview

7.3 UC3: Anomaly Detection in Manufacturing Settings

Companies that manufacture expensive high added-value parts are very demanding regarding machine availability and quality assurance. However, some techniques that are used require the machine to stop before performing the analysis.

This UC proposes a deployment approach where data analysis is performed continuously while the hardware equipment keeps running most of the time, and the state of the various independent components, along with the overall status, is continuously reported. Moreover, the UC will focus on a single critical component, ball screws. Ball screws are expensive and critical machine components whose breakage implies stopping the machine for a significant time.

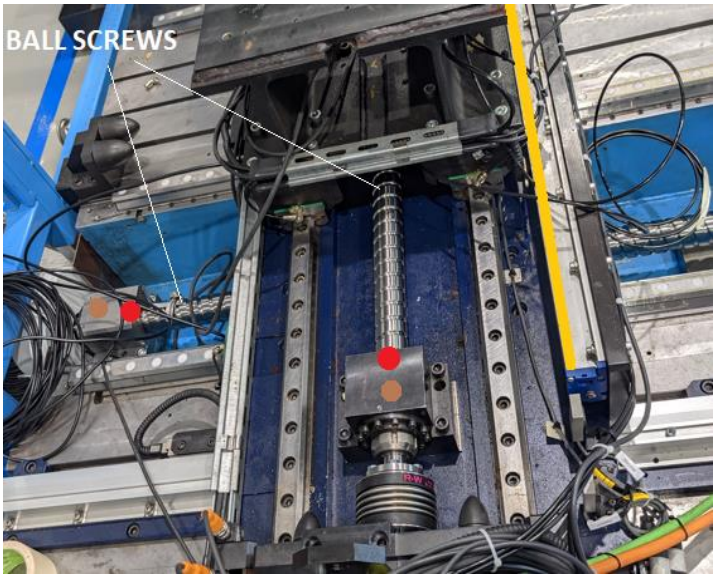


Figure 37: Test bench – simulating a machine with two ball screws in the X and Y axes to generate the movement of the machine. These ball screws are sensorized (brown and red points) with position and acceleration sensors

The UC is developing a Data Processing Application to analyse real-time signals from the ball-screw sensors and check for anomalies, detecting anomalous behaviours that may affect the part quality and predict imminent failures. This application has been divided into two different services that analyse the data coming from the position sensors and the data from the acceleration sensors (Position Processor service and Acceleration Processor service) of the ball screw. In addition, to obtain data from real machines at Ideko's facilities, a test bench has been built with two sensorized ball screws (Figure 37), simulating data from machines in a real scenario.

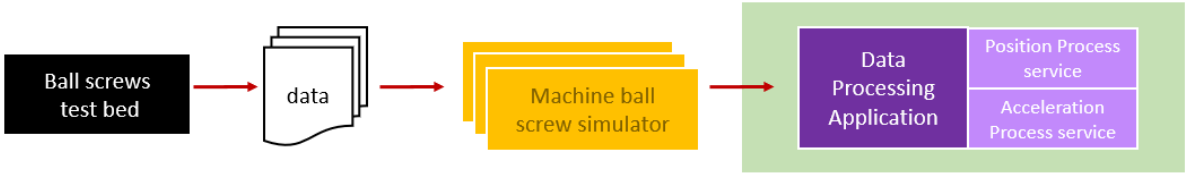


Figure 38: Data from ball screw to Data Processing Application services sequence simulating data from machines in a real scenario

The workflow is as follows, data are being monitored by a mechanism running on the edge device attached to the machine, which monitors every single parameter of the machine. When new data are found, the Stream Processor Connector sends it to a Stream Data Processing Tool layer (in the context of SERRANO, this can be the Data Broker). Next, the Stream Processor Connector sends the data to a Data Processing Application, where the appropriate processor service is triggered depending on the nature of the data.

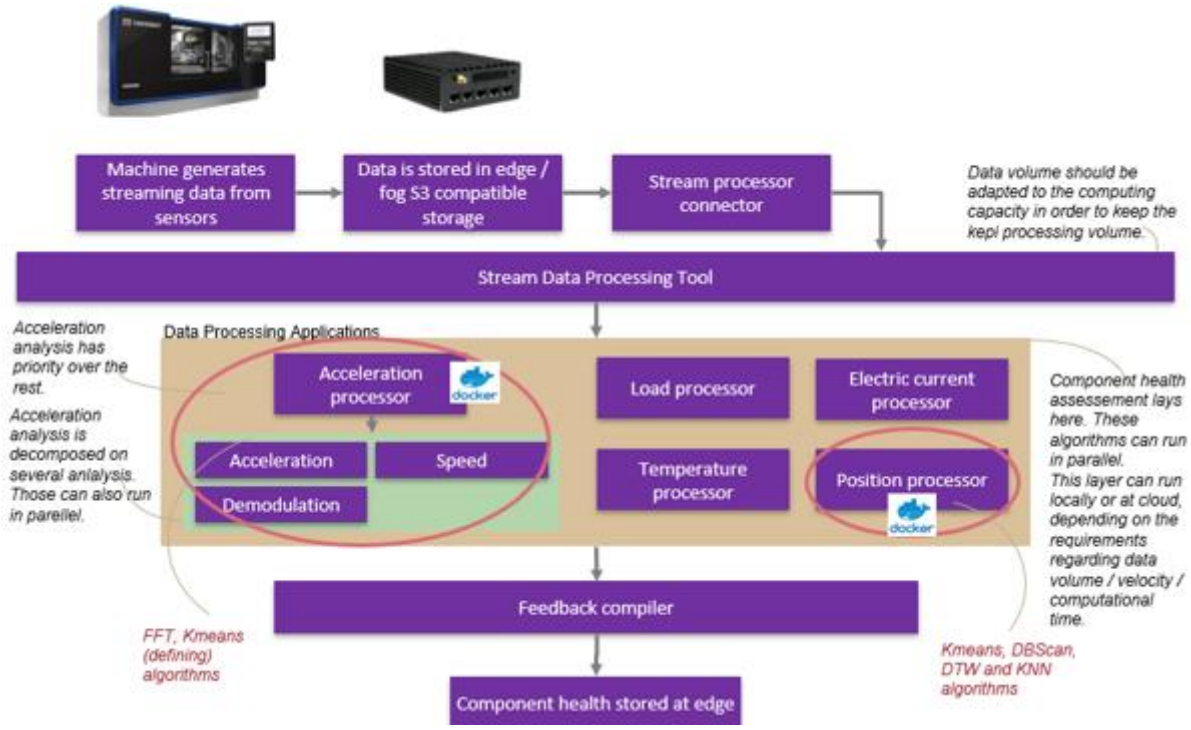


Figure 39: Use case workflow

In order to execute these developed services and obtain the ball-screws' status assessment without stopping the machine in real-time scenarios, this use case will take advantage of the components provided by the SERRANO platform. That will ensure service quality in terms of latency, constantly adapting to the current demand of resources. On the one hand, since the algorithms used in these services (Kmeans/DBScan/DTW/FFT/KNN) for clustering raw data and classifying real-time data are computationally demanding, data cannot always be processed at the edge (the computing resources at the edge are limited). On the other hand, the edge is not able to use a high volume of data streaming to make predictions using the trained classifier in order to perform anomaly detection.

For these reasons, SERRANO will accelerate the execution of the proposed algorithms through the components of the acceleration abstraction mechanisms (FPGA-enabled HW acceleration / GPU-enabled HW acceleration). In addition, when the classifier is retrained, these data will be stored (Secure Storage Service) to be processed at high speed, giving the possibility of processing them in Cloud/HPC systems. The idea is to reduce the classifier retraining time and the time needed to make a new prediction using the streaming data. This way, we can avoid possible imminent failures of the ball screws and achieve greater control of the health status of the ball screw in real-time.

Moreover, as previously mentioned, the use case will utilize the capabilities of the Data Broker component through the exposed publish-subscribe interface to forward the data generated by the use case machines to the SERRANO platform services (Stream Data Processing Tool in the Workflow figure). For this purpose, a connector will be created in the machine ball screw simulator application that will publish data in the SERRANO Data Broker component, from where the applications/services/components will consume the data using the appropriate subscriptions.

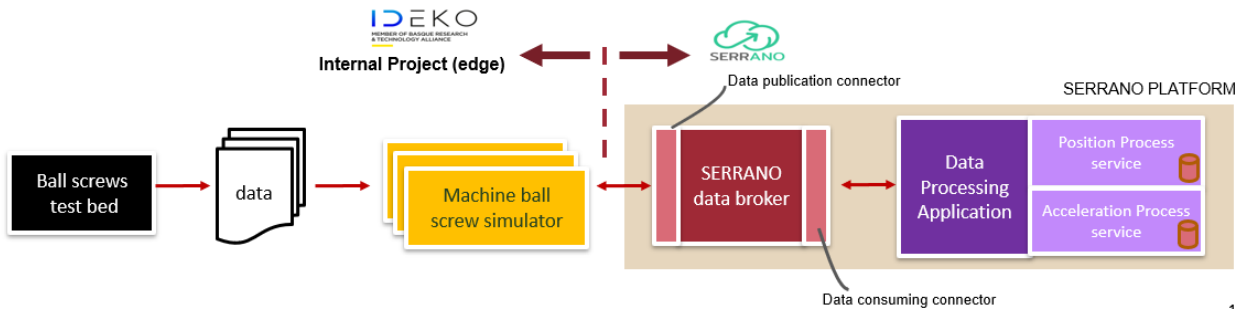


Figure 40: Transfer data from machine ball screw simulator application to SERRANO using Data Broker.

Furthermore, to facilitate the transparent application implementation and management into the SERRANO platform, the appropriate description of its requirements (application intent) will be exposed. This way, the SERRANO orchestration mechanisms can allocate the platform resources to satisfy the application's workload requirements.

8 SERRANO Implementation and Delivery Plan

8.1 Software Engineering Approach

A crucial part of complex software systems development and delivery lifecycle is the Continuous Integration (CI) and Continuous Delivery/Deployment (CD) – jointly mentioned as CI/CD. CI, in software development, is a practice of building/integrating and testing all developer working code frequently in a shared code repository. A common CI practice is integrating the changed code at least daily. The frequent integration helps the contributors to notice any arising errors and correct them instantly. A more detailed description of the software engineering approach can be found in deliverable D6.3 "The SERRANO integrated platform" (M18).

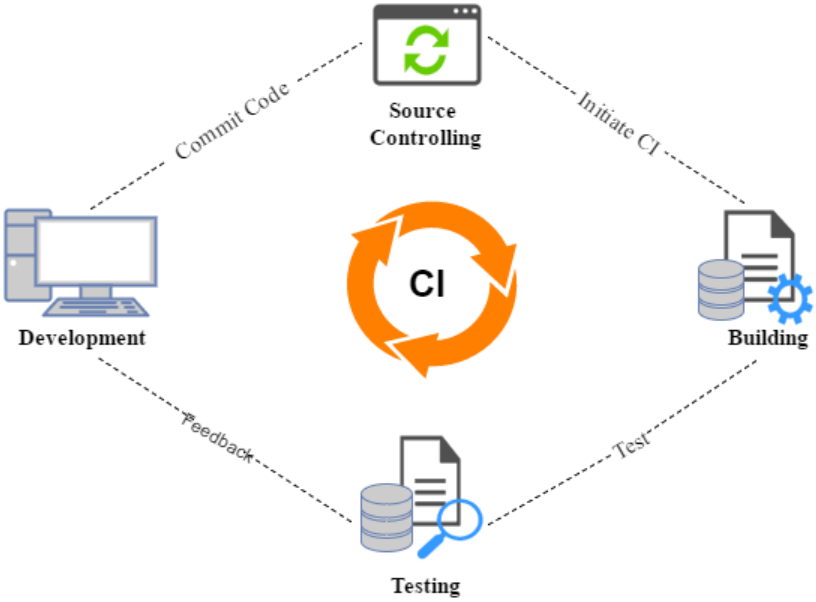


Figure 41: The Continuous integration lifecycle

Continuous Integration offers significant advantages such as:

- reduction of risk in the development, since it reveals any possible incompatibility between software components early during the development phase,
- facility of instant bug fixing,
- availability of the current product version at any moment, as the code is frequently integrated.

As INTRA will be the system integrator, to support integration and system testing activities, INTRA will capitalize on CI/CD platform, composed of tools that support the entire software lifecycle processes up to the release and deployment of fully tested operational systems.

The CI/CD environment consists of the following tools:

- **GitLab** for source control and tracking, code repository, code versioning
- **Jenkins or Gitlab CI/CD** for automated building, testing, and deployment
- **Docker** for bundling the developed services and components into containers using de facto standards
- **SonarQube** for performing static analysis of code to detect bugs, code smells, and security vulnerabilities. Dependency check plugin for SonarQube can dependencies as well for security vulnerabilities
- **Harbor** for managing, storing, and distributing the produced binary files (Docker images) and checking Docker image scanning results. **Trivy** or another tool can be used to scan the image for vulnerabilities
- **DefectDojo** for efficiently managing vulnerabilities
- **NGINX** for efficiently managing requests towards the deployed services

The typical development workflow to be followed by the technical partners, as well as the association of the different development processes to the CI/CD tools, is depicted in the following figure.

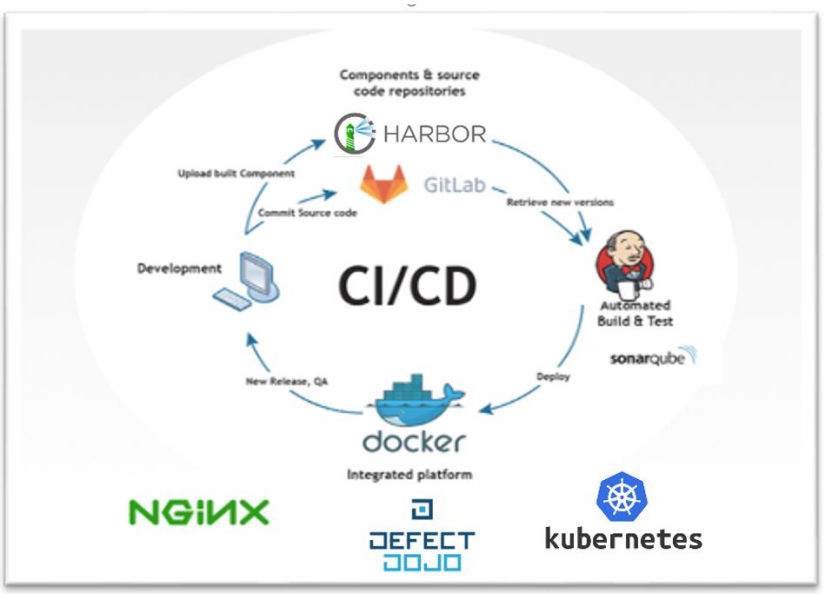


Figure 42: CI/CD tools in development workflow

As an example, a partner could follow these steps to successfully complete the workflow:

- Step 1: Develop component
- Step 2: Create OpenAPI YAML or JSON and validate on https://editor.swagger.io/
- Step 3: Commit and push code and Swagger to Gitlab https://gitlab.com/serranoproject
- Step 4: Check Unit Test execution and below reports on Jenkins

- Step 5: Check SonarQube report for security vulnerabilities on code and dependencies
- Step 6: Check Integration Tests execution
- Step 7: Build new docker image
- Step 8: Check DefectDojo/Harbor for image scanning report coming from Trivy
- Step 9: Deploy image to integration environment
- Step 10: Run service health tests in integration environment

8.2 Implementation Schedule

The overall work within the SERRANO project is organized based on a set of well-defined and complementary phases (Figure 43) that start with the definition of the use cases and the requirement analysis (Phase 1). Next, SERRANO adopts an iterative approach for the definition of the architecture (Phase 2), the implementation and evaluation of the individual technological developments (Phase 3), and the overall platform integration (Phase 4). The design and implementation activities will be implemented according to a spiral model with two iterations (M01-M18, M19-M36), each of which will include a series of activities that bridge the gap between requirement analysis, technology, and innovation. Since the WP2 is completed at M18, this deliverable also concludes the related activities corresponding to Phases 1 and 2. Moreover, the outcomes will be the base of the second iteration of the implementation and integration. According to the integration plan, the developed components and services will be continuously integrated with the defined interfaces and communication protocols as set in the SERRANO architecture specification. Finally, the project will conclude with the demonstration of the UCs (Phase 5) and the exploitation of the results (Phase 6).

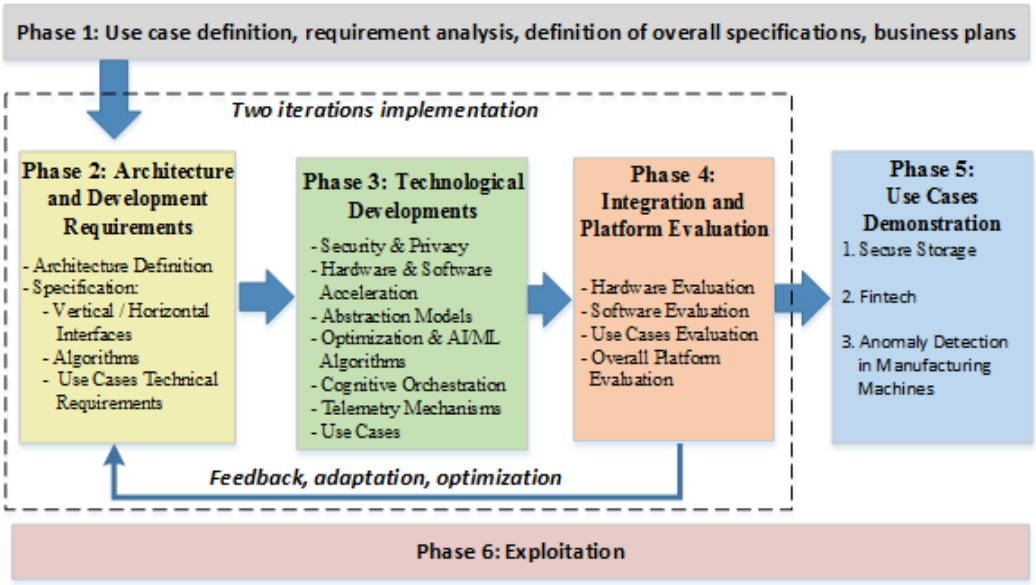


Figure 43: Overall technical development strategy and methodology

Based on the above development strategy, the three main releases (Figure 44) for the integrated SERRANO platform are:

- Initial platform prototype in M18
- Full platform prototype in M33
- Final platform prototype in M36

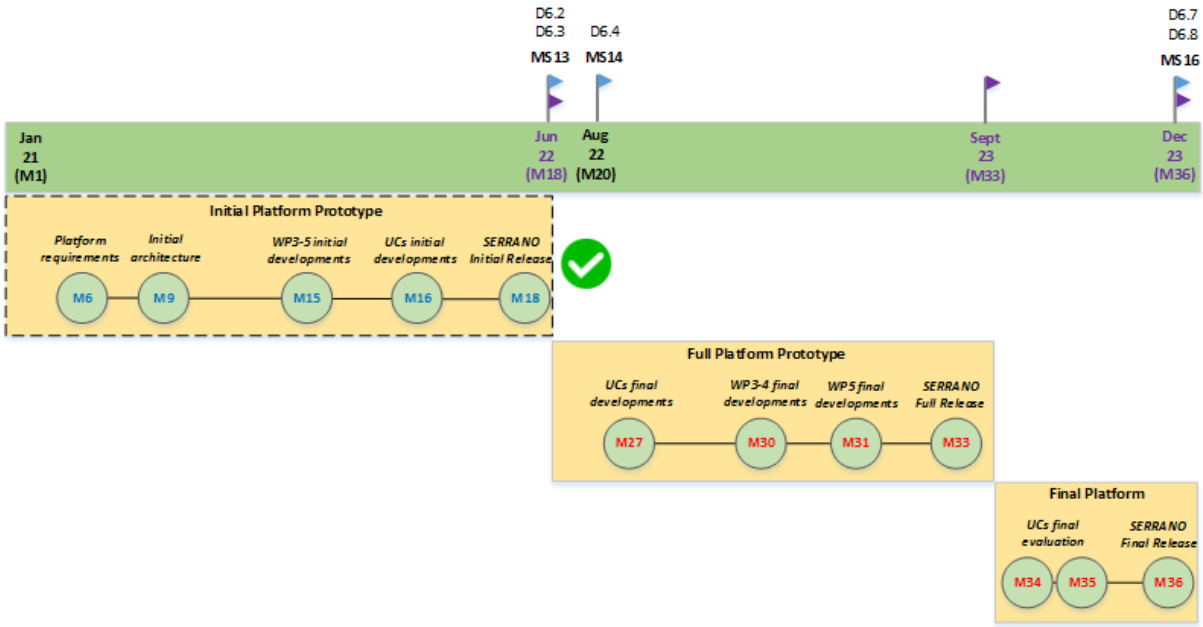


Figure 44: SERRANO development roadmap

The initial platform prototype is the outcome of the first development iteration and will provide the partial implementation of SERRANO components. In this version, each component implements a subset of the envisioned features along with the primary interfaces for inter-component communication. The initial release aims to provide a basic functional prototype that will support the core functionalities of the three project use cases. Deliverable D6.3 “The SERRANO integrated platform” (M18) provides its description and documentation, while the analysis and feedback of the initial integration activities were used to finalize the SERRANO architecture.

The SERRANO complete platform prototype will be based on the initial release and provide the remaining functionality, which has not been included in the early prototype. As the development in technical work packages (WP3 – WP5) concludes by M31, this release will be provided in M33. The intention is to achieve a fully functional platform that integrates all project components and provides a prototype suitable for the pilots’ experimentation.

For the final release of the SERRANO platform, delivered at the end of the project, the consortium will focus on implementing the feedback from the final evaluation of the SERRANO platform (Phase 5) through the demonstration of the three project use cases. The outcomes from the demonstrators will be collected and documented, and videos will be prepared and

uploaded to project communication and dissemination channels. This version will be fully integrated and documented as part of deliverables D6.7 “Final version of SERRANO integrated platform” (M36) and D6.8 “Final version of business, end user and technical evaluation” (M36). Moreover, it will contribute to the identification of the critical and high-impact components in order to create the final exploitation plans.

9 Requirements Coverage

The following table presents how the final set of functional requirements, collected, described, and categorized in D2.4 "Final version of SERRANO use case, platform requirements and KPIs analysis" (M16) are served by the components of the SERRANO architecture. During the overall SERRANO architecture design, all partners closely collaborated to analyse each requirement and map it to the platform's components, following an iterative and collaborative interaction among the technical and UC partners of the project. The goal was to ensure that the SERRANO platform could realize the desired functionalities.

Table 39: Functional requirements served by the SERRANO architecture

ID	Short Description	Priority	Component
F_GR.1	Provide a unified view of cloud, edge and HPC resources	Core	Central Telemetry Handler, Enhanced Telemetry Agents, Monitoring Probes, ARDIA models
F_GR.2	Enable an intent-driven paradigm of federated infrastructures	Core	AI-enhanced Service Orchestrator, Resource Orchestrator ARDIA models, SERRANO SDK
F_GR.3	Support transparent application deployment	Core	AI-enhanced Service Orchestrator, Resource Orchestrator ARDIA models, SERRANO SDK
F_GR.4	Encompass an autonomous and continuous control loop	Core	Resource Orchestrator, Central Telemetry Handler, Orchestration Drivers, Runtime Controller, Enhanced Telemetry Agents
F_GR.5	Support safety-critical, latency-sensitive and data-intensive applications	Core	AI-enhanced Service Orchestrator, Resource Orchestrator, Distributed Secure Storage, On-premises Gateway, DPU HW accelerated encryption, HW acceleration, Trust Execution and Lightweight Virtualization Mechanisms
F_GR.6	Expose well-defined APIs through SERRANO SDK	Core	SERRANO SDK
F_GR.7	Support of additional application areas	Desired	AI-enhanced Service Orchestrator, ARDIA models, SERRANO SDK
F_ECHAR.5	Application error tolerations	Desired	Hardware and Software Approximate Kernels, Run-time Controller for Approximate Kernels, ARDIA models
F_ECHAR.6	Device selection in design time	Essential	Run-time Controller for Approximate Kernels
F_ECHAR.7	Accelerated kernels integration	Essential	Run-time Controller for Approximate Kernels, Hardware Acceleration Abstractions, ARDIA models

F_ECHAR.9	Implement accelerate-able kernels as vAccel plugins	Desired	Hardware Acceleration Abstractions
F_SIR.1	Authentication, encryption, privacy and data integrity of communications	Core	DPU Hardware Accelerated Encryption, Distributed Secure Storage
F_SIR.2	Secure storage service should support most common S3 operations	Core	Distributed Secure Storage, On-premises Gateway, Edge Storage
F_SIR.3	Automatic creation/ selection of storage policies	Essential	Distributed Secure Storage, On-premises Gateway, Resource Optimization Toolkit
F_SIR.4	Secure storage service should maintain data access from browsers	Desired	On-premises Gateway
F_SIR.5	Secure execution of workloads	Essential	Trusted Execution Mechanisms
NF_SIR.1.upd	Secure storage service should provide low latency file access	Essential	Edge Storage, On-premises Gateway
NF_SIR.2	Secure storage service should enforce end to end encryption	Essential	DPU Hardware Accelerated Encryption, Distributed Secure Storage, On-premises Gateway
F_ NCTFR.1	Discover and monitor heterogeneous resources	Core	Central Telemetry Handler, Enhanced Telemetry Agents, Monitoring Probes, ARDIA models
F_ NCTFR.2.upd	Dynamically monitor short-lived applications (support serverless architecture)	Core	Hardware Acceleration Abstraction, Monitoring Probes, Enhanced Telemetry Handler
F_ NCTFR.3.upd	Estimate inter-site and intra-site network characteristics	Core	Enhanced Telemetry Agents, Monitoring Probes, Central Telemetry Handler
F_ NCTFR.4.upd	Autonomously monitor cloud-native applications over heterogeneous distributed resources	Core	Central Telemetry Handler, Enhanced Telemetry Agents, Orchestration Drivers, Resource Orchestrator
F_ NCTFR.5	Detect critical situations that may lead to application reconfigurations or data migration	Core	Enhanced Telemetry Agents, Service Assurance and Remediation
F_ NCTFR.6	Exchange events between the components of the hierarchical telemetry infrastructure	Core	Enhanced Telemetry Agents, Data Broker
F_ NCTFR.7.upd	Zero-touch and data-driven reconfigurability of telemetry components	Core	Central Telemetry Handler, Enhanced Telemetry Agents, Orchestration Drivers
F_ NCTFR.8.upd	Ability to centralize the monitoring information in a common view or place	Core	Central Telemetry Handler, Enhanced Telemetry Agents, Persistent Monitoring Data Storage
F_ NCTFR.9	Low-level metrics availability	Essential	Lightweight Virtualization, HW Acceleration Abstractions, Enhanced

			Telemetry Agents, Monitoring Probes
F_ROSAR.1	Support cognitive and multi-object resource allocation algorithms	Core	Resource Optimization Toolkit, Energy and Resource Aware Mapping, Uncertainties Estimation Framework
F_ROSAR.2	Support seamless and declarative orchestration of self-organized distributed orchestration systems	Core	Resource Orchestration, Orchestration Drivers, ARDIA models
F_ROSAR.3.upd	Ability to coordinate workload migration	Core	Service Assurance, Resource Orchestration, Orchestration Drivers, Distributed Secure Storage
F_ROSAR.4	Support automatic data migration operations	Core	Service Assurance, Resource Orchestration, Orchestration Drivers, Distributed Secure Storage
F_ROSAR.5	Orchestrate deployment of data segments over distributed edge and cloud resources	Core	Resource Optimization Toolkit, Orchestration Drivers, Distributed Secure Storage
F_ROSAR.6	Ability to implement custom scheduling policies	Core	Orchestration Interface, Orchestration Plug-ins
F_ROSAR.7	Time-based ordering of monitoring data entries	Core	Enhanced Telemetry Agent, Data Broker
F_ROSAR.8	Persistent storage of monitoring data	Core	Central Telemetry Handler
F_ROSAR.9	Data formatting and pre-processing	Core	Data Broker
F_ROSAR.10	Access to the real time monitoring stream bus	Core	Central Telemetry Handler, Enhanced Telemetry Agent, Data Broker
F_ROSAR.11	Transprecise adaptation ML methods	Core	Service Assurance and Remediation Enhanced, Telemetry Agents
F_ROSAR.12	Transprecise Hyper-parameter optimization methods	Core	Service Assurance and Remediation
F_ROSAR.13.new	Deployment description of applications should include their task decomposition and technical details.	Core	AI-enhanced Service Orchestrator, ARDIA models
F_SOR.1.upd	Decomposition of applications into independent well-defined tasks	Core	AI-enhanced Service Orchestrator, ARDIA models
F_SOR.2.upd	Description of tasks should support additional information (i.e. metadata, deployment requirements)	Core	ARDIA models
F_SOR.3.upd	Accurate description of heterogenous resources	Core	ARDIA models

	capabilities in a machine processable format		
F_SOR.4.upd	Platform should infer the appropriate resource characteristics for each application micro-service	Core	AI-enhanced Service Orchestrator, Central Telemetry Handler, ARDIA models
F_SOR.5	Platform should be able to predict potential service requirements based on data-driven mechanisms	Core	AI-enhanced Service Orchestrator, Central Telemetry Handler, Service Assurance Mechanisms
F_SOR.6	Ability to specify specific security and privacy requirements regarding applications and data	Core	ARDIA models, Distributed Secure Storage, Trust Execution and Workload Isolation Mechanisms
F_SOR.7	SERRANO cognitive orchestration mechanisms as a service	Core	AI-enhanced Service Orchestrator, Resource Orchestrator, SERRANO SDK
F_SOR.8.new	Expression of collected telemetry data in predefined format	Core	ARDIA models, Monitoring Probes, Central Telemetry Hander, Persistent Monitoring Data Storages
F_SOR.9.new	Parameters prioritisation in applications' description	Core	ARDIA models, AI-enhanced Service Orchestrator

10 Conclusions

This deliverable reports on the work performed in WP2 regarding the definition of the final version of SERRANO platform architecture. The proposed architecture enables the implementation of the SERRANO overall vision to introduce a cognitive abstraction layer that transforms the distributed, secure and accelerated edge, cloud, and HPC resources into a single borderless infrastructure.

The deliverable highlights the SERRANO-enhanced hardware and software resources and the primary services/components of the SERRANO platform. A more technical presentation of the SERRANO advancements is available at the respective technical deliverables from Work Packages 3-5. The deliverable builds on top of the initial architecture specification, as presented in D2.3 (M9), to enhance the multi-layer overall architecture of the SERRANO platform while extending the description of the information flows between the SERRANO components and the vertical and horizontal components' interfaces. The updated development roadmap of the SERRANO platform is also presented. Moreover, the document presents how the final architecture serves the challenging technical and use case-related requirements, whose initial and final descriptions are presented at D2.2 (M6) and D2.4 (M16).

This document will be further used as a guideline, during the second iteration of the implementation, for the technical activities in WPs 3-5 and UCs development and platform integration activities in WP6. Hence, D2.5 will also ensure that the outcomes of WP3-6 are relevant and aligned with the SERRANO ambition to introduce a novel ecosystem of cloud-based technologies, spanning from specialized hardware resources up to software toolsets that will enable the transparent application deployment in a secure, accelerated and cognitive cloud continuum.

11 References

[1] HLRS SYSTEMS 2016-2020: https://www.hlrs.de/systems/

[2] The List Top500, https://www.top500.org

[3] B. Dick, T. Bönisch, B. Krischok, Hawk Interconnect Network, HLRS, (2020), https://kb.hlrs.de/platforms/upload/Interconnect.pdf

[4] H. R. Zohouri, A. Podobas und S. Matsuoka. “High-Performance High-Order Stencil Computation on FPGAs Using OpenCL”. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). 2018, S. 123–130. url: https://arxiv.org/pdf/2002.05983.pdf

[5] NVIDIA Jetson Nano - https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[6] Canonical MAAS – Metal as a Service - https://maas.io/

[7] Canonical JUJU - https://jaas.ai/

[8] K3s Lightweight Kubernetes - https://k3s.io/

[9] FastAPI benchmarks: https://www.techempower.com/benchmarks/#section=test&runid=7464e520-0dc2-473d-bd34-dbdfd7e85911&hw=ph&test=query&l=zijzen-7

[10] ASGI: https://asgi.readthedocs.io/en/latest/

[11] Kodo: https://www.steinwurf.com/kodo

[12] D. Williams, R. Koller, M. Lucina, N. Prakash, “Unikernels as Processes”, SoCC '18: Proceedings of the ACM Symposium on Cloud Computing, October 2018. url: https://dl.acm.org/doi/10.1145/3267809.3267845XX

[13] Kubernetes: https://kubernetes.io

[14] Slurm workload manager: https://slurm.schedmd.com/documentation.html

[15] TORQUE resource manager: https://adaptivecomputing.com/cherry-services/torque-resource-manager/

[16] OpenPBS OpenPBS: https://www.openpbs.org/

[17] Apache Kafka: https://kafka.apache.org

[18] RabbitMQ: https://www.rabbitmq.com

[19] Quobyte, "CASE STUDY EFFICIENT OPERATIONS AND FAST DATA ACCESSENABLE BOTTLENECK-FREE RESARCH AT HLRS", url: https://www.quobyte.com/case-studies/hlrs

[20] MinIO S3 Gateway: https://docs.min.io/docs/minio-gateway-for-s3.html

[21] Prometheus Query: https://prometheus.io/docs/prometheus/latest/querying/basics/