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

Grant Agreement no. 101017168

Deliverable D2.3
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:	30/09/2021
Actual submission date:	30/09/2021

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
11.04.21	ICSS	Draft	0.1	Initial ToC
01.06.21	ICSS	Draft	0.2	Draft ToC circulated within the entire consortium
05.07.21	IDEKO	Draft	0.3	Initial content provided in Section 3.1.4
08.07.21	INTRA	Draft	0.3	Initial content provided in Sections 3.3.5 and 7.1
12.07.21	AUTH	Draft	0.3	Initial content provided in Sections 3.1.1 and 3.3.3
16.07.21	ICCS	Draft	0.4	Initial content provided in sections 3.3.2, 3.3.4, 3.3.5, 4.1, 4.2 and 7.2
20.07.21	CC	Draft	0.5	Initial content in Sections 3.1.4 and 3.2.1
23.07.21	HLRS	Draft	0.5	Initial content provided in Section 3.1.3
26.07.21	MLNX	Draft	0.6	Initial content provided in Section 3.1.2
26.07.21	AUTH	Draft	0.7	Extend content in Sections 3.1.1 and 3.3.3
27.07.21	UVT	Draft	0.8	Initial content in Sections 3.3.3, 3.1.4
27.07.21	INNOV	Draft	0.9	Initial content provided in Section 3.3.1, Section 4.2 and Section 5
30.07.21	INTRA	Draft	0.9	Initial content provided in Section 5, additional content in Section 3.3.5
05.08.21	ICCS	Draft	0.10	Integrate content by INTRA, INB, NBFC, HLRS in Sections 3.1.4.3, 3.2.2, 3.2.3, 3.2.4, 3.3.2.3, 3.3.3.4 and 7.1
11.08.21	CC	Draft	0.11	Content added to Sections 3.2.1, 4.2.5, 5.1 and 6.1
23.08.21	IDEKO	Draft	0.12	Initial content provided in Section 6.3
27.08.21	ICCS	Draft	0.13	Add content by HLRS in Sections 3.1.3, 4.2. Extend Sections 2, 3.3.2 and 8
30.08.21	AUTH	Draft	0.14	Content added in section 4.2.3
01.09.21	ICCS	Draft	0.15	Updates and comments on Section 3
07.09.21	ICCS	Draft	0.16	Integrate content by HLRS, INTRA and UVT in Section 3. Extend Section 4.2
15.09.21	ICCS	Draft	0.17	Integrate content by CC, INB, INTRA, AUTH, UVT, HLRS, NBFC
20.09.21	ICCS	Draft	0.18	Finalize all deliverable sections
23.09.21	ICCS	Draft	0.19	Revised version before internal review
29.09.21	ICCS	Draft	0.20	Integrated post review contributions
30.09.21	ICCS	Final	1.0	Final version for submission.

Author List

Organization	Author
ICCS	Aristotelis Kretsis, Panagiotis Kokkinos, Polyzois Soumplis, Emmanouel Varvarigos
MLNX	Yoray Zack, Juan Jose Vegas Olmos
CC	Marton Sipos
USTUTT/HLRS	Dmitry Khabi, Javad Fadaie Ghotbi
AUTH	Dimosthenis Masouros, Argyris Kokkinis, Kostas Siozios
INTRA	Paraskevas Bourgos, Makis Karadimas
INB	Javier Castillo, Ferad Zyulkyarov
INNOV	Filia Filippou, Vassiliki Andronikou, Efstathios Karanastasis, Efthymios Chondrogiannis
IDEKO	Aitor Fernández, Javier Martín
UVT	Silviu Panica, Iuhasz Gabriel
NBFC	Anastasios Nanos

Internal Reviewers

Paraskevas Bourgos and Makis Karadimas, INTRA

Babis Chalios, NBFC

Abstract: This deliverable (D2.3) presents the outcomes of *Task 2.3 "Architecture Specification"*, *Work Package 2 "Requirements and System Design"* of the SERRANO project, during the first iteration of the incremental implementation plan. The deliverable presents the initial architecture specifications of the SERRANO platform. It also provides a breakdown of the platform's key building blocks and describes their interactions along with the required interfaces. Moreover, the SERRANO implementation and delivery plan is introduced.

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

Disclaimer: *The information, documentation and figures available in this deliverable are written by the SERRANO Consortium partners under EC co-financing (project H2020-ICT-101017168) and do not necessarily reflect the view of the European Commission. The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The reader uses the information at his/her sole risk and liability.*

Table of Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
ASGI	Asynchronous Server Gateway Interface
ASIC	Application-Specific Integrated Circuit
BSI	Belief Desire Intention
CFD	Computational Fluid Dynamics
CI/CD	Continuous integration/Continuous Deployment
CNC	Computer Numerical Control
CORS	Cross-Origin Resource Sharing
DOCA	Data center On a Chip Architecture
DPU	Data Processing Unit
DSE	Design Space Exploration
EDE	Event Detection Engine
EU	European Union
FaaS	Function as a Service
FPGA	Field Programmable Gate Array
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
MAAS	Metal as a Service
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
RTL	Register-Transfer Level
SAR	Service Assurance and Remediation
SDK	Software Development Kit
SFTP	Secure File Transfer Protocol

SLA	Service Level Agreement
SoC	System on a Chip
SW	Software
TLS	Transport Layer Security
TPM	Trusted Platform Module
TTM	Time to Market
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 in the edge, enabling accelerated edge nodes as integral parts of the computation and storage chain. 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 D2.3 “SERRANO Architecture” provides an initial reference architecture of the SERRANO platform. The innovations and technological advancements of the SERRANO project, are also highlighted providing detailed information on the novel ecosystem of cloud-based technologies, spanning from specialized hardware resources up to software toolsets that the SERRANO utilizes and develops.

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. The SERRANO architecture will be finalized in M18 in D2.5 “Final version of SERRANO architecture”.

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.

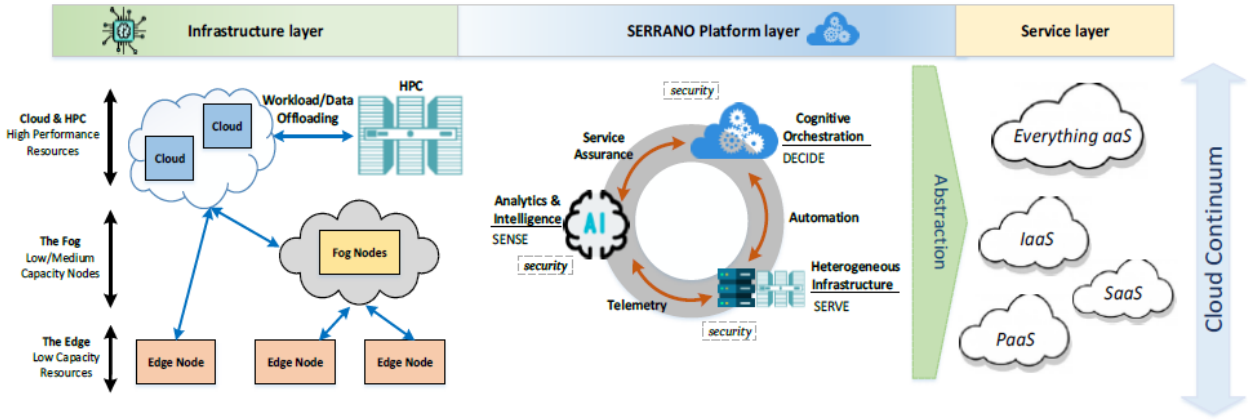


Figure 1: The SERRANO platform, utilizing edge, cloud and HPC resources and empowering the Everything as a Service (EaaS) notion towards the cloud continuum

2.1 Purpose of this document

The present deliverable (D2.3) presents the outcomes of Task 2.3 – "Concept, Use Case Requirements and Business Prospects" of Work Package 2 – "Requirements and System Design of the SERRANO project", during its first iteration (M1 – M18) of the 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.

The objective of D2.3 is to build on the initial contributions of Tasks 2.1 and 2.2, as reported in the respective deliverables in M6 (D2.1 and D2.2), towards the provision of a fundamental description of SERRANO's hardware and software components and the associated technical specifications for the next steps of the project. Also, D2.3 presents the high-level architecture of the SERRANO platform, the interactions among the platform's key building blocks and the associated interfaces. This deliverable also presents a preliminary description of the SERRANO platform deployments that will support the project's use cases (UCs).

D2.3 will serve as a guideline for each research and development activity conducted in the technical work packages (WP3-5) and the UCs development, evaluation, and integration activities under WP6. To this end, each technical work package will have to ensure that project's developments are fully aligned with the initial architecture and interfaces. At the same time, as the technical work evolves, the architecture will be modified incrementally with more technical details into the various components and interfaces. Finally, an updated version of this document entitled D2.5 – "Final version of SERRANO architecture" will be provided at the end of the first iteration of the project design and implementation activities (M18).

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 of use to anyone interested in the initial description of the SERRANO components, the specification of the overall SERRANO architecture and the preliminary interfaces. Moreover, this document can be also 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 that correspond 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

Hardware accelerators are dedicated acceleration systems that aim to increase the efficiency of computations in modern general-purpose processors such as CPUs. Most routines or functions that can be computed in parallel can be calculated in software running on a generic CPU but also purely in custom-made hardware. A wide range of algorithms can be offloaded onto a hardware accelerator throughout a variety of fields such as Machine Learning, Computer Vision, Video editing/rendering, Digital signal processing, Cryptography or even Finance. However, making use of hardware acceleration makes sense when computationally intensive algorithms need to be accelerated. Accelerators are not appropriate for performing most of the processing in everyday computing which is often serial in nature or has many conditional branching. In these cases, a CPU might be a better option as it often runs in higher clock frequencies than a typical accelerator core and can be more optimal in serial tasks.

Nevertheless, delegating performance-critical functions to specialized external hardware is often a good tactic to reduce the execution time of a program or increase the energy efficiency. The most common hardware used for acceleration include Graphics Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs). These devices can handle large amounts of data in many streams, performing

relatively simple operations on them in parallel. GPUs are devices with a large number of specialized processors while FPGAs and ASICs implement fixed-function algorithms that run purely in the fabric. These devices can be found in edge computing where they operate close to the source of the data in the form of a small computer (with embedded memory, CPU, IOs, etc.) or in the cloud usually attached in a server.

GPUs and FPGAs are hardware devices usually attached in a PCIe slot or packaged in a complete System on a Chip (SoC). In the two tables that follow (Table 1 and Table 2), we describe detailed specifications for both PCIe and Edge acceleration devices.

Table 1: Hardware edge devices lineup - Detailed specs

Device name	**ZCU102** 	**ZCU104** 	**Xavier NX** 	**Xavier AGX** 
CPU chip	quad-core ARM Cortex-A53+ dual-core Cortex-R5F	quad-core ARM Cortex-A53+ dual-core Cortex-R5F	6-core NVIDIA Carmel ARM®v8.2	8-core ARM v8.2
GPU	Mali-400 MP2	Mali-400 MP2	384-core Volta GPU+Tensor cores	512-core Volta GPU+Tensor cores
Memory Capacity	4GB 64-bit	2GB 64-bit	8 GB 128-bit	32GB 256-Bit
Memory Bandwidth	20 GB/s	20 GB/s	51.2 GB/s	137 GB/s
Camera Interfaces	USB 3.0	USB 3.0	2x MIPI CSI-2 DPHY lanes	(16x) CSI-2 lanes
Display Interface	64 GB	HDMI and display port	HDMI and display port	HDMI 2.0
PCI Express	PCIe Gen2	PCIe Gen2	PCIe Gen3	PCIe Gen3
DSP Slices	2520	1728	-	-
Maximum Power	15W	15W	15W	30W

Table 2: Hardware PCIe devices lineup - Detailed specs

Device name	Alveo U200 	Alveo U50 
Width	Dual Slot	Single Slot
Thermal Cooling	Passive	Passive
PCI Express	Gen3x16	Gen3x16, 2x Gen4
DDR Total Capacity	64 GB	-
DDR Total Bandwidth	77 GB/s	-
HBM Total Capacity	-	8 GB
HBM Total Bandwidth	-	460 GB/s
Network Interface	2x QSFP28	1x QSFP28
DSP Slices	6840	5952
Maximum Power	225W	75W

3.1.1.1 Programmability of FPGA accelerators

Designing hardware for FPGAs can be performed at varying levels of abstraction with the commonly used being the register-transfer level (RTL) and the algorithmic level. These methodologies differ; RTL (i.e., VHDL, Verilog) is used for circuit-level programming while algorithmic level methodologies such as High-Level Synthesis (HLS) are used for describing designs in a more abstract way. Figure 3 and Figure 4 depict the designing steps of the aforementioned methodologies. Compared to RTL, HLS provides a faster and more flexible development process, as designers instruct the HLS compiler on how to synthesize accelerators by adding different directives on a C/C++ or OpenCL code.

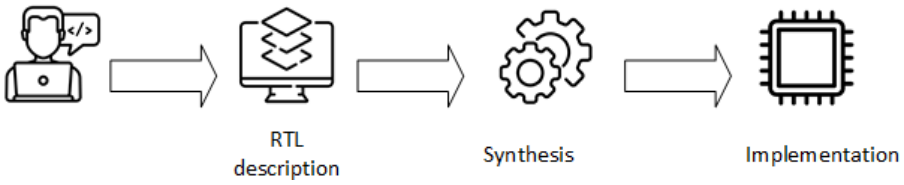


Figure 3: Register-Transfer Level Methodology

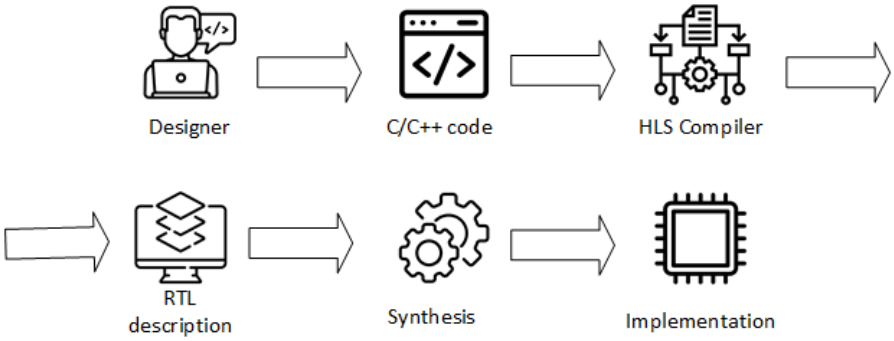


Figure 4: High-Level Synthesis Methodology

Xilinx Inc., one of the main FPGA vendors on the market, has put significant effort into providing a user-friendly toolset for employing HLS for FPGA design. In this context, Xilinx introduced Vitis, a framework that provides a unified OpenCL interface for programming edge (e.g., MPSoC ZCU104) and cloud (e.g., Alveo U200) Xilinx FPGAs. Vitis aims to simplify the designing process and as a result let developers focus on the Design Space Exploration (DSE) phase, which targets the performance optimizations with respect to the architecture and resources of the available devices.

3.1.1.2 Programmability of GPU accelerators

On the other hand, GPUs' programmability has been significantly improved over the past years. High level languages (such as CUDA and OpenCL) target the GPUs directly, so GPU programming is more and more becoming mainstream in the scientific community.

For the purposes of SERRANO project, we make use of NVIDIA GPUs and therefore the CUDA programming model is used. There are three key language extensions CUDA programmers can use: CUDA blocks, shared memory, and synchronization barriers. CUDA blocks contain a collection of threads which can share a specific type of memory called shared memory, while they can pause until all threads reach a specified set of execution.

The CUDA programming model enables developers to scale software, increasing the number of GPU processor cores as needed, while we can use CUDA language abstractions to program applications and divide programs into small independent problems. In a more technical manner, GPU programs are written in *.cu* files. CUDA C adds the *__global__* qualifier to standard C in order to alert the NVIDIA (*nvcc*) compiler that a function, called kernel, should be compiled to run on a device instead of the host. Developer can pass parameters to a kernel as they would with any C function after having determined the kernel grid *(<<<grdiDim, blockDim>>>*). Note that *gridDim* refers to the number of blocks, where *blockDim* to the number of threads per block. Additionally, and before kernel execution, developers need to allocate memory in the GPU's global memory using *cudaMalloc()*. This call behaves very similarly to the standard C call *malloc(),* but it tells the CUDA runtime to allocate the memory on the device.

Finally, the CUDA developer accesses memory on a device through calls to *cudaMemcpy()* from host code. These calls behave exactly like standard C *memcpy()* with an additional parameter to specify which of the source and destination pointers point to device memory.

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.

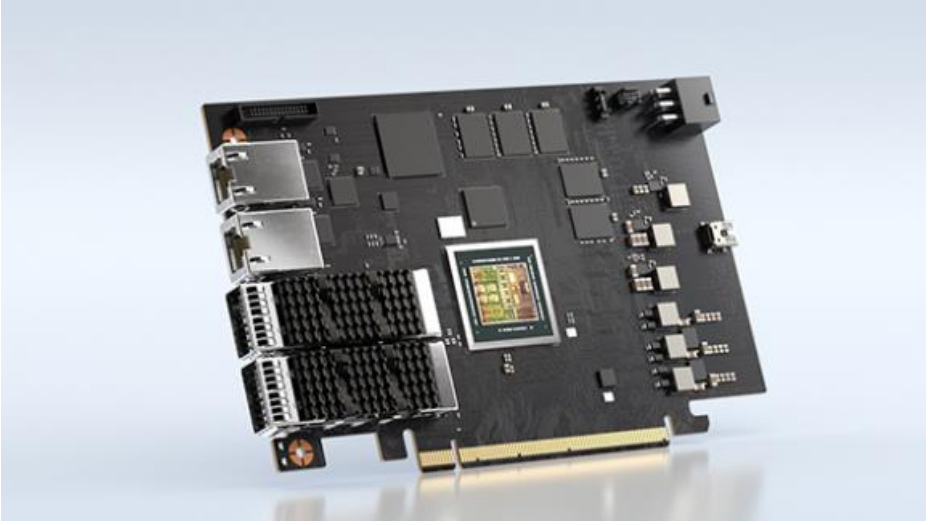


Figure 5: BlueField2 DPU: 8 ARM A72 CPUs, 8MB L2 cache, 6MB L3 cache in 4 Tiles, ARM Frequency: 2.0-2.5GHz, Dual 10-100Gb/s Ethernet & InfiniBand single 200Gb/s, PCIe gen4.

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 wide range of the HPC applications. The HPE Apollo 9000 Hawk system is the most powerful of them. At time of installation, it among the fastest high-performance computers in the world and the fastest general-purpose machine for industrial production in Europe. In Toop500'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 an efficient use of the supercomputers. This is an important condition for the integration of the HPC services in SERRANO project and imposes certain requirements on the use cases of the project and the SERRANO Orchestrator.

Figure 6 on the right side shows a schematic representation of 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 switches of a rack 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 9th dimension is not fully implemented. For the complete implementation of a 9D-hyper cube the number of nodes has to be 8192. The implementation of Hawk's interconnect required 3,024 cables with a total length of 20 km.

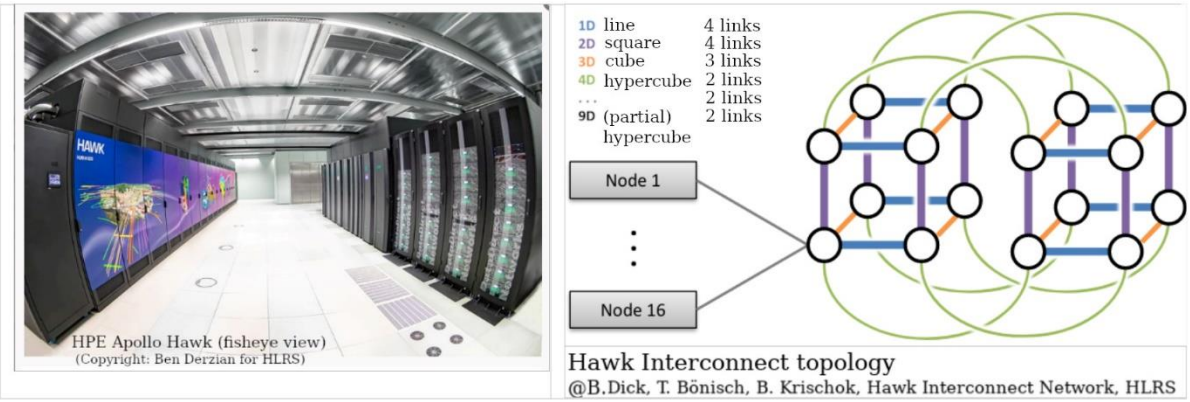


Figure 6: On the left, 20 of totally 44 cabinets of the supercomputer Hawk.. On the right, part of 9D-Hypercube internetwork topology graph (4D hypercube).

Due to the high 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 take into account 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 3 describes the main components of the Hawk.

Table 3: 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 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 on Figure 7 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 [5]. 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 two versions for the newer “Intel Skylake” or “AMD EPYC Rome” processors.

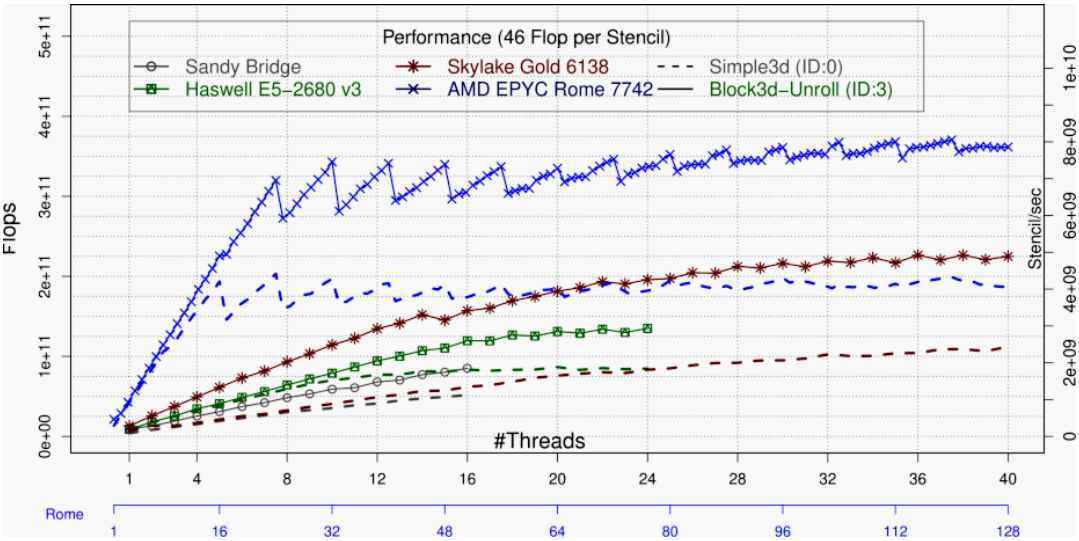


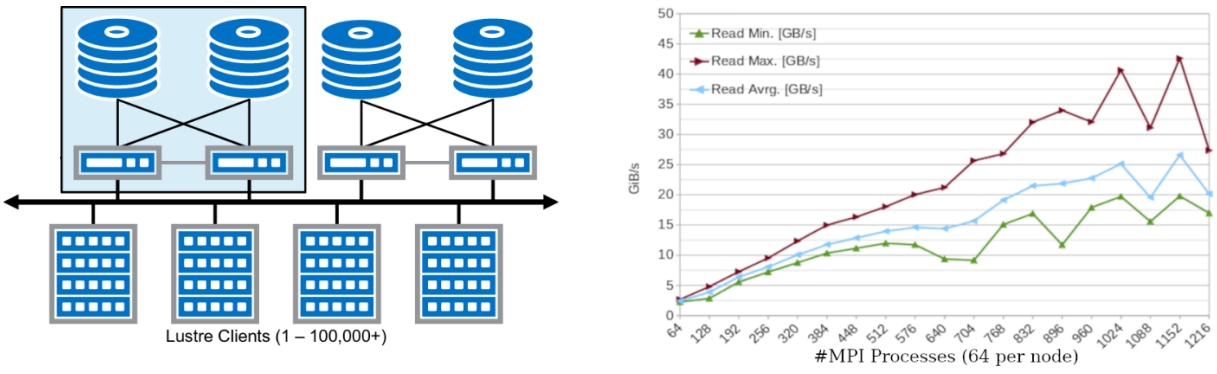
Figure 7: Performance of the different implementations of the 3D Laplace- operator on dual systems

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 8 shows a scheme of a Lustre file system (www.lustre.org). The file system architecture allows parallel access to the multiple raids of the hard disks in order 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 from 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.



(a) (b)

Figure 8: (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).

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].

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 9 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.

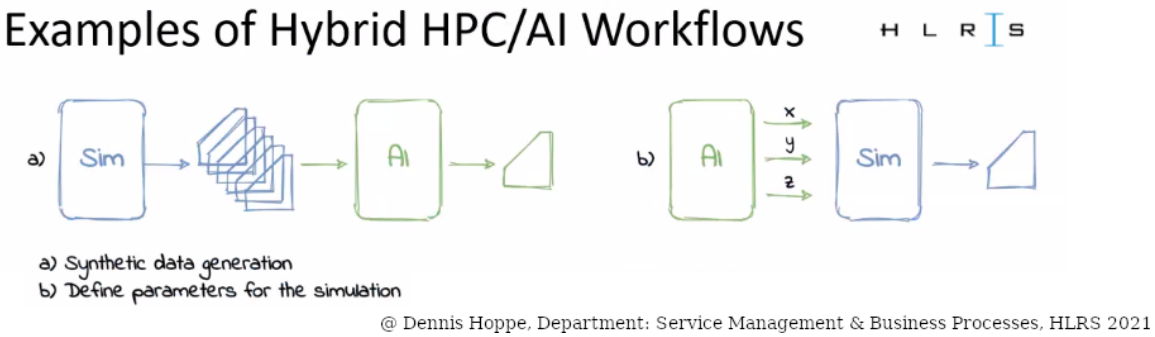


Figure 9: Examples of Hybrid HPC/AI Workflows.

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 comes already with all the connectivity needs 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, 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-premise storage gateway and edge resources

The On-premise 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. We will also investigate the possibility of accelerating encryption and erasure coding through the use of FPGAs and GPUs, described in Section 3.1.1. If successful, this will lead to further CPU offloading as well as 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-premise 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-premise 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 SERRANO project's 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 docker 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

To facilitate the deployment of workloads on the SERRANO platform, virtualization is a key component. 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. For testing and development purposes UVT is setting up a mini-EDGE devices cluster (custom made, presented in Figure 11) using NVIDIA Jetson Nano [6] development boards. 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).

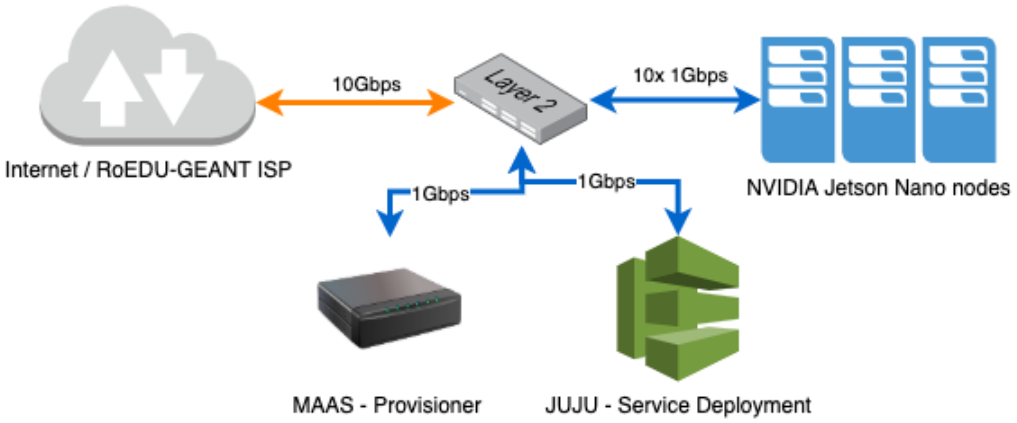


Figure 10: UVT Jetson Nano Cluster Setup

Each Jetson Nano devices is setup to boot from the attached SSD drive and it has an 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 10 the cluster setup is depicted. The Jetson Nano devices are administered by a MAAS [7] bare metal provisioning service that controls and bootstraps the basic operating system on the device hard drive. Further, the JUJU [8] 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 [9], 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 for creating multiple individual mini-clusters to accommodate various testing and development edge-like environments.

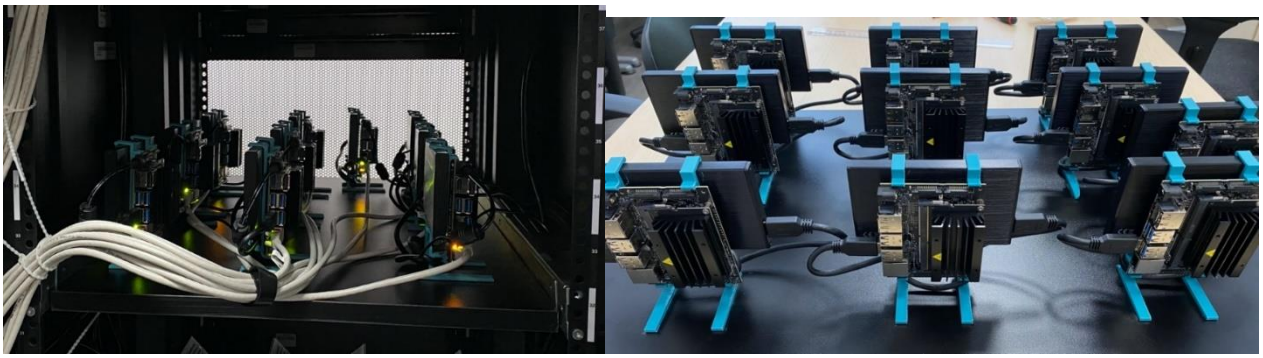


Figure 11: UVT Jeston Nano Cluster - Rack Mount

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 2.2 includes a detailed presentation of the service's main 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 3rd parties and curious cloud providers. Thanks to its multi-cloud approach, the system is able to 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 that stretches to the edge. It does this by offering edge storage locations on SERRANO edge devices and an On-premise storage gateway specifically tailored to enterprise users. 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 D2.2) and hardware acceleration possibilities as described in Section 3.1.4.2. A high-level view of the service's components is presented on Figure 12.

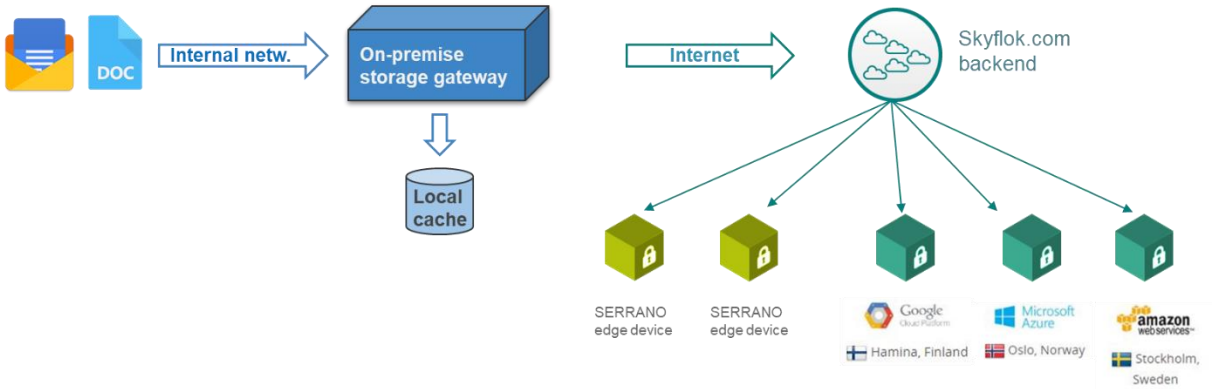


Figure 12: The components of the SERRANO-enhanced Storage Service

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, it is likely some changes will be needed, including a new API designed to provide information about the state and general characteristics of the cloud storage locations.

3.2.1.1 On-premise storage gateway

The On-premise storage gateway (Gateway from now on) will be the central 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 3.7 or newer. It will be built on either Flask 2.0 with the async module enabled or FastAPI. Both web frameworks provide a way to reliably serve a large number of HTTP requests simultaneously. Flask is the more established solution with good support and a large community of developers that use and extend it. 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 [10] 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) [11]. 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 significant time, efficient asynchronous I/O is crucial to achieving a good user experience. Chocolate Cloud will assess both options through deep-dive prototypes. While at first it appears, the Gateway should be built using FastAPI, a decision will be made after the prototypes have been evaluated.

The Gateway will feature data processing in two notable ways: encryption and erasure coding. 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 perhaps less know technique. As such, we will either use a CPython-based wrapper around Kodo [12], a high-performance C++ implementation or develop our own Python-native implementation. The decision will be made based on several factors including performance, potential for acceleration, licensing and so on.

To enable simple and flexible deployment, the Gateway will be a containerized application. It will maintain 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-premise storage locations where SERRANO users can have their files distributed to. At a high level, this component should be fairly simple, it must provide a simple storage API though 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 full 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 uses 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 generally 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 13 illustrates how a file is downloaded using a pre-signed URL. Furthermore, to maintain browser-based access, support for Cross-Origin Resource Sharing (CORS) is also needed.

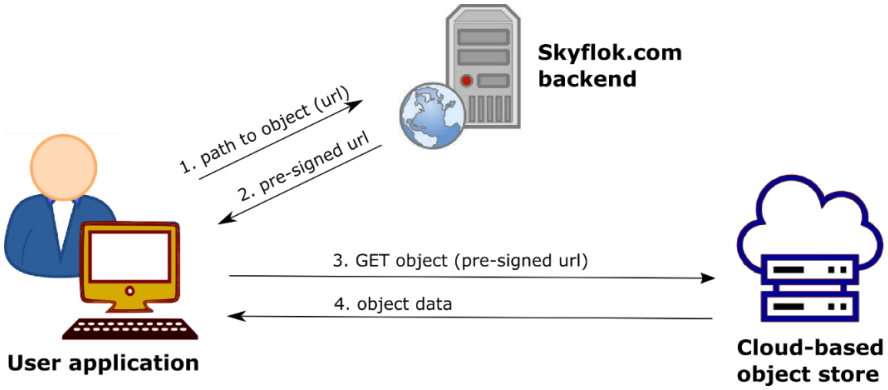


Figure 13: Using a pre-signed URL to download a file

Given the aforementioned requirements and those presented in D2.2, the choice is between using an off-the-shelf object store and a custom implementation. We have assessed three different object stores: Ceph, OpenStack Swift and MinIO. Of the three, we favour MinIO, a relatively 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.

In terms of security, it is simple to set up access key pairs used by the Skyflok.com backend to authenticate itself with MinIO deployments. This 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. 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 a good 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 possibly 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.

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 [13] 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, 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 SGX/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 to the management layer raising at the same time security concerns. In 5G-COMPLETE[1], we introduce 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, we introduce vAccel, a virtual acceleration framework tailored for cloud-native, lightweight virtualization setups.

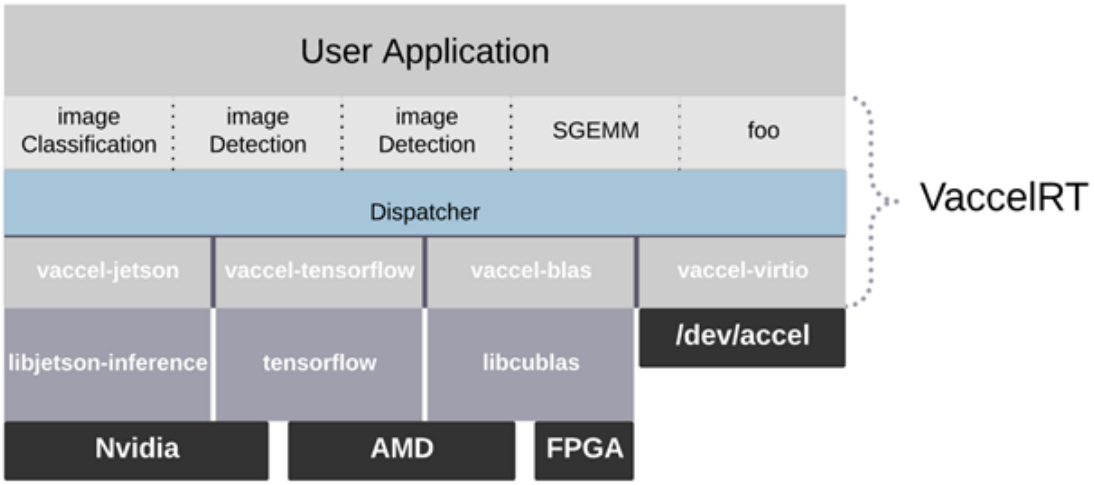


Figure 14: vAccel Framework

vAccel, depicted in Figure 14, 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 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.

3.2.4 Lightweight virtualization for seamless deployment

The Cloud computing paradigm appears ideal to deploy and manage application execution at scale. To support the cloud computing execution model at the edge, devices need to support virtualization and run a full hypervisor stack.

Using container technology as an alternative to the full virtualization stack at the Edge in order to increase performance, but keep interoperability, incurs significant security concerns. To this

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

end, SERRANO introduces a hybrid deployment model where lightweight virtualization mechanisms are used in combination 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 users of the platform, 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 are able to support VM and container execution, as well as unikernel deployment on a single system.

The system software that is being developed for the SERRANO platform encompasses the above technologies to form a unified, secure & 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 & 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.

4 SERRANO Platform Components

SERRANO adopts a lifecycle methodology (Figure 15) 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 the decomposition of the high-level requirements into specific service goals (*step b*). Then, the actual 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 they trigger any required re-optimization.

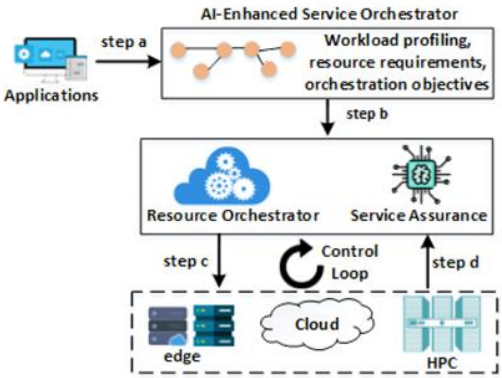


Figure 15: Lifecycle workflow for SERRANO applications

In what follows, we present the SERRANO components.

4.1 AI-enhanced Service Orchestrator

4.1.1 ARDIA Framework

The parameters of interest for the execution of the UC applications through the SERRANO platform should be specified in advance through a series of Abstraction Models that will be developed. The Abstraction Models will be available in a predefined format so that it can be consumed by the other components of the SERRANO platform. Since these will define the semantics and structure for the applications', services' and resources' needs in the context of SERRANO, they will also guide 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 taking into account the overall goal of the end user, the possible deployment scenarios and the collected SERRANO telemetry data.

Table 4: 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 and services 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 - 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.

Next, we present the internal components of the ARDIA Framework and the interaction of this component with the other entities of the SERRANO platform.

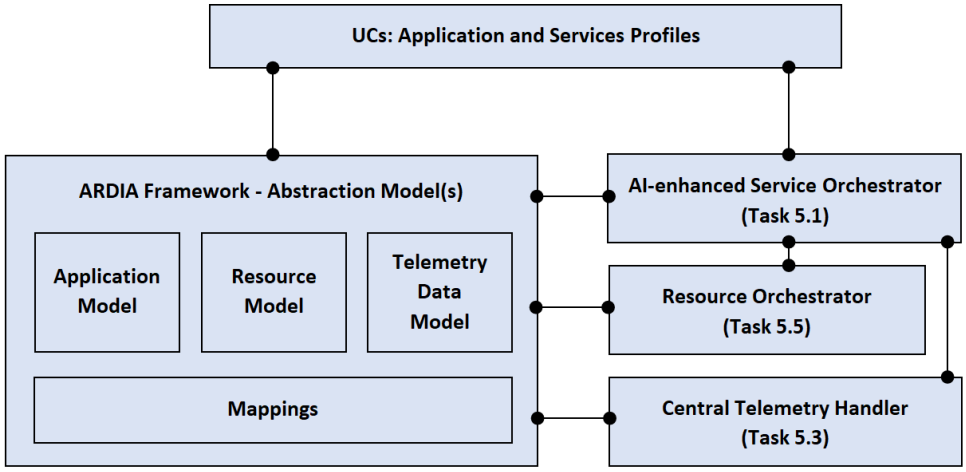


Figure 16: ARDIA Framework components and dependencies

Table 5: 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 its translation purposes (internal) and its interface with the Telemetry API - Central Telemetry Handler for expressing any application-level information (both for capturing and communication)
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 6: 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 available at 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 7: 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 both physical and software resources) including their overall status and communication data. This model may include both 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 particular resource

	- Telemetry sub-system for expressing measured parameters - Local Orchestrators - SERRANO Infrastructure (edge devices, cloud providers) for expressing relevant data
UCs	The Telemetry Data Model provides also 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 taking into account the metadata provided about each particular application, the parameters being necessary for the deployment purposes of the application by the Resource Orchestrator, the infrastructure usage related data collected so far from the Central Telemetry Handler and the overall goal of the end user (aka intent) at that time. All the aforementioned data should be expressed based on the elements of the Abstraction Model(s) of the ARDIA framework presented in Section 4.1.1.

Table 8: Description of AI-enhanced Service Orchestrator

Component ID	WP5T1AISO
Name	AI-enhanced Service Orchestrator
High level description	This component is responsible for analysing the applications profiles and translate their high-level requirements into specific infrastructure related operational constraints and orchestration objectives by taking into consideration both the application and services requirements and the forecasting of their 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, 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 the following figure.

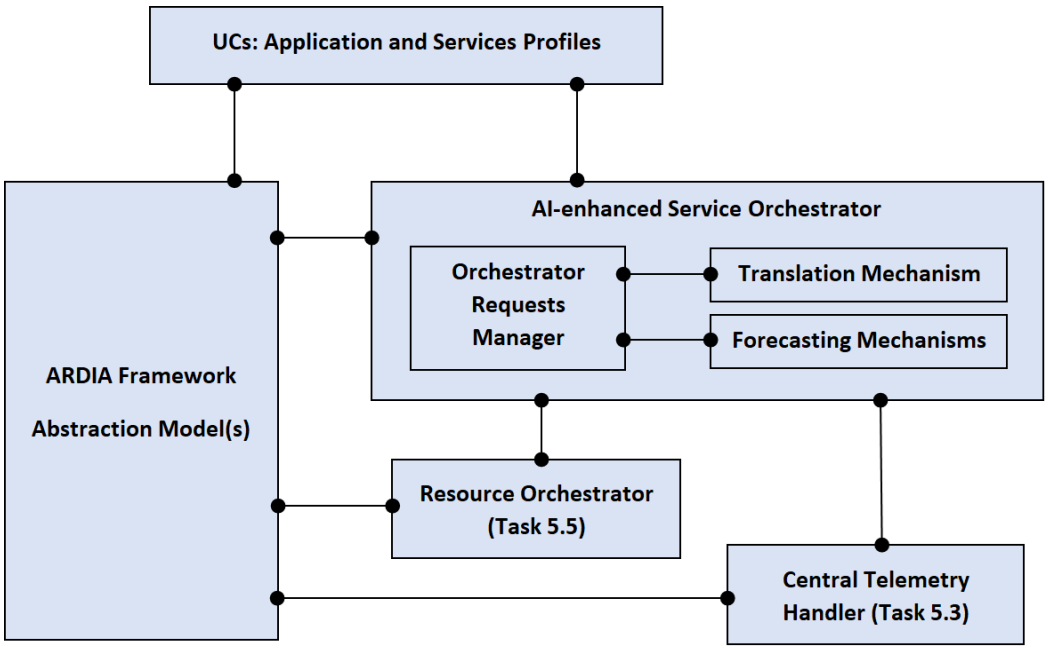


Figure 17: AI-enhanced Service Orchestrator core components and dependencies

Table 9: 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	- Forecasting Mechanisms (for AI-enhanced Service Orchestrator) for receiving requests and providing forecasted expected needs for informing the Requests Manager - Translation Mechanism is being fed with the specified UC application requirements, goals and the forecasted needs of the applications, services and tasks - 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 10: Description of Translation Mechanism

Component ID	WP5T1TM
Name	Translation Mechanism
High level description	It is responsible for analysing the applications profiles and translating their high-level requirements into specific infrastructure related operational constraints and orchestration objectives.
Collaborators	- ARDIA Framework is used for expressing the application, service, task needs and providing the mappings among the different models - Orchestrator Requests Manager is triggered to feed the component with 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 11: 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	- Orchestrator Requests Manager fed with the forecasting results for the Service Orchestrator to enrich/adjust its output - Central Telemetry Handler is used for collecting relevant UC application data from the SERRANO infrastructure - 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 to support the transparent application deployment over the 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). SERRANO Resource Orchestrator adopts a declarative approach, instead of an imperative one, for describing the workload requirements to the selected Local Orchestrators. This provides several degrees of freedom to each Local Orchestrator for serving in an optimal manner the "request", satisfying both the central orchestrator and the resource's objectives.

SERRANO considers that Local Orchestrators are based on existing and well-established solutions, like container orchestration mechanisms (e.g. Kubernetes [14], Swarm [15]) for edge and cloud resources and specific schedulers or meta-schedulers (e.g. Slurm [16], TORQUE [17]) for HPC platforms. For implementing the proposed hierarchical resource orchestration, SERRANO will also provide 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, implemented as plugins, will 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.

The core building blocks of the SERRANO Resource Orchestrator and Orchestration Drivers along with their interactions with other SERRANO components are presented in Figure 18.

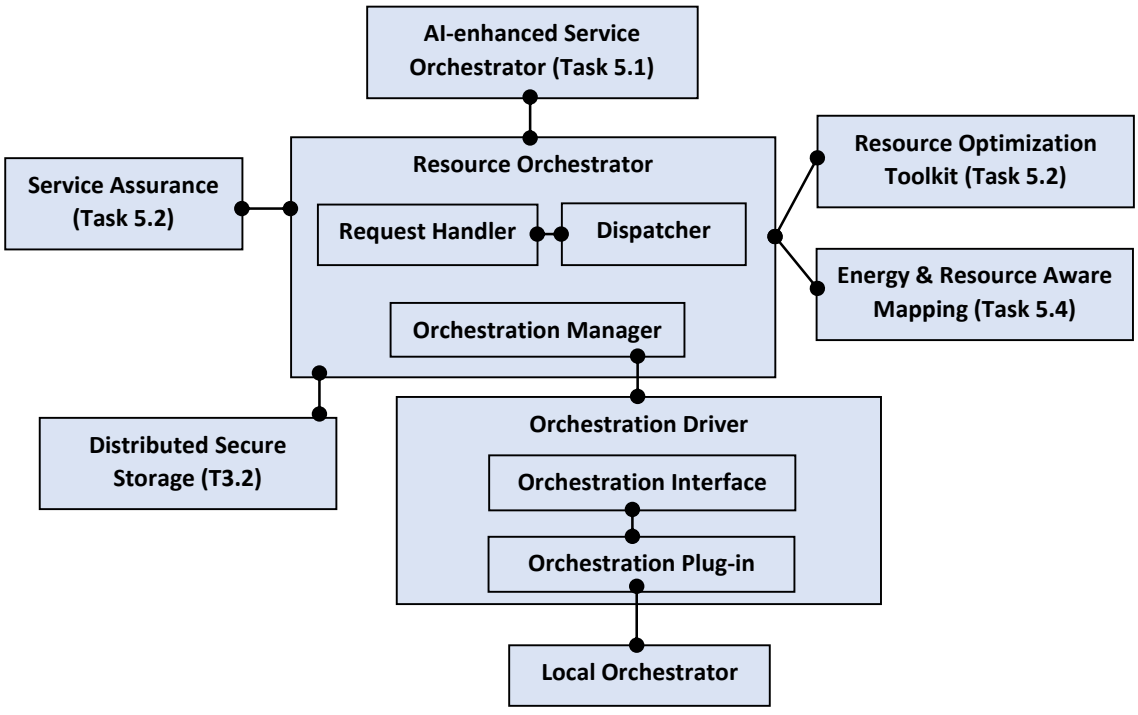


Figure 18: Resource Orchestrator and Orchestration Drivers core components and dependencies

Table 12 provides the initial description of these components, whose detailed design and development will be performed within Task 5.5. More technical details will be included at deliverables D5.3 “Resource orchestration, telemetry and lightweight virtualization mechanisms” (M15) and D5.4 “Intelligent service and resource orchestration mechanisms” (M31).

Table 12: Description of resource orchestrator and orchestration drivers core components

Component ID	WP5T5RO
Name	Resource Orchestrator
High level description	It will ensure efficient service orchestration and resource management in the disaggregated and heterogeneous SERRANO infrastructure. Taking as input from the AI-enhanced Service Orchestrator the application 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 individual tasks constraints, while it also initiates the application deployment and automatically coordinates the necessary supplemental actions (e.g. transfer required data).
Collaborators	- ARDIA Framework (WP5T1AF) - AI-enhanced Service Orchestrator (WP5T1AISO) - Resource Optimization Toolkit (WP5T2ROT) - 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	WP5T5RH
Name	Request Handler
High level description	It will handle the interaction with the AI-enhanced Service Orchestrator and the other high-level SERRANO components (e.g. Service Assurance). It will act as the single-entry point for the other components to SERRANO resource orchestration functionalities. Initially, the Request Handler will validate the requests and then forward them to the Dispatcher.
Collaborators	- ARDIA Framework (WP5T1AF) - AI-enhanced Service Orchestrator (WP5T1AISO) - Dispatcher (WP5T5D) - 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 is the main component of the Resource Orchestrator as it implements the application logic, oversees the operation of the other internal components and coordinates the resource allocation and application deployment operations.
Collaborators	- ARDIA Framework (WP5T1AF) - Request Handler (WP5T5RH) - Resource Optimization Toolkit (WP5T2ROT) - Energy & Resource Aware Mapping (WP5T4EMT) - Orchestration Driver (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 will prepare the required application deployment instructions based on the Resource Optimization Toolkit's (ROT) decisions and the characteristics of the selected Local Orchestrators. Moreover, it will coordinate the required data movement through the interaction with the Distributed Secure Storage component and trigger the application deployment by interacting with the Orchestration Drivers at the selected edge/cloud/HPC platforms.
Collaborators	- Dispatcher (WP5T5D) - Distributed Secure Storage - 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 will provide an abstraction layer for the interaction with the specific edge, cloud, HPC orchestration mechanisms at each SERRANO individual location. It will receive by the Resource Orchestrator, through the Orchestration Manager, requests for deploying or adjusting already deployed applications. SERRANO adopts a declarative approach for expressing the instructions to the Local Orchestrators.
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 will provide at the Orchestration Manager and Orchestration Driver an infrastructure agnostic interface for descripting the deployment preferences and constraints to the heterogeneous local orchestration mechanisms. It will also handle the interaction with the SERRANO Distributed Secure Storage service.
Collaborators	- Orchestration Manager (WP5T5OM) - Orchestration Plug-ins (WP5T5OP) - Distributed Secure Storage
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 will translate the generic instructions from the Orchestration Interface to specific actions and procedures for the selected local orchestration mechanisms. The interaction will be based on the API and tools exposed by each local orchestration component. There will be 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 challenges for management and service deployment. 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 will provide to the SERRANO Resource Orchestrator the required logic to allocate the edge, cloud and HPC resources so as to satisfy the applications' requirements, to coordinate the efficient movement of required data across the selected resources and to support proactively and reactively re-optimization adjustments.

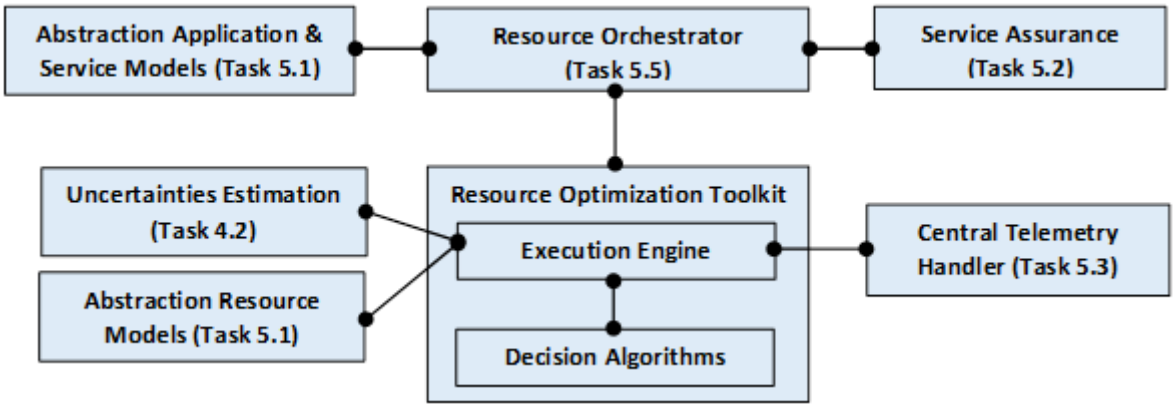


Figure 19: Resource Optimization Toolkit core components and dependencies

Figure 19 illustrates the key building blocks of the ROT, its main components and the interactions with other SERRANO components and Table 13 provides a high-level description of these components. Task 5.2 is responsible for the development of the resource allocation algorithms, the detailed design and implementation of the ROT. More technical details will be provided at deliverables D5.2 "Algorithmic framework, performance and power models" (M15) and D5.4 "Intelligent service and resource orchestration mechanisms" (M31).

Table 13: 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. It is responsible for the preparation, coordination, and management of 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	WP5T2EE
Name	Execution Engine
High level description	It receives requests, for starting or terminating algorithm executions, through the common interface from the Resource Orchestrator and performs all the required actions, including the preparation of the execution environment, monitoring progress and forwarding the results to Resource Orchestrator.

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) - Resource Abstraction Models, provide details for the resource requirements (WP5T1AM) - Decision Algorithms (WP5T2DA)
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	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 the energy-efficiency. Hence, the energy-aware design, configuration and execution of the UC applications have to be supported by the SERRANO platform. This relates particularly to the provided HPC services.

As long as a UC application meets the requirements for running in a HPC environment, such as high-level degree of parallelization, 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 a 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 not only 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, but also provide information about their behaviour depending on the particular 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.

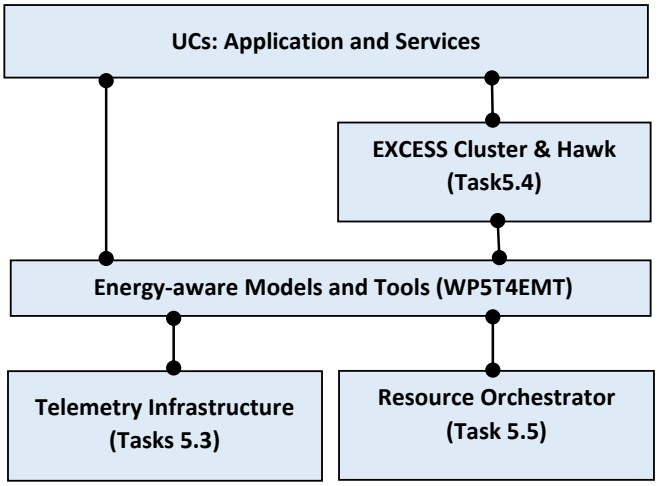


Figure 20: Energy and resource aware mapping core components and dependencies

Table 14: 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 taken into consideration 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 results of the previous step (see description of component WP5T4EXCESS) we will extend the existing power and performance models, adapting them to the specific features of the considered hardware (NUMA topology, memory/cache bandwidth and latencies, CPU-frequencies, network parameters). The proposed models will enable developers to observe schematically the execution of applications and services 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 set of concurrent tasks (see D2.2 section 5.2.5 Requirements for efficient use of supercomputers). The models will help to identify the optimal execution strategies for the execution of 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 a very complex task, due to the great 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. The SERRANO platform will provide several service assurance mechanisms based on artificial intelligence and machine learning techniques, that will facilitate the autonomous adaptation and management of the deployed services and resources. Specifically, SERRANO aims to provide such mechanisms across different levels, ranging from hardware-aware tuning techniques up to application-specific optimizations.

4.3.1 Device-aware application mapping on GPU and FPGA accelerators

SERRANO aims not only to accelerate applications, but to study and apply several device-specific code optimizations in order to fine-tune them in terms of both performance and power efficiency. Therefore, in order to achieve applications’ close-to-peak efficiency, several optimization techniques are applied for both FPGA and GPU accelerators in order to efficiently map applications to the corresponding hardware resources. In this way, the SERRANO

platform exploits the full potentials of accelerators in terms of both performance and/or energy, thus, enabling QoS and SLA guarantees of deployed applications

4.3.1.1 Code optimizations for fine-tuned GPU accelerated applications

As far as GPUs are concerned, we examine resource management techniques from a software perspective to improve latency and power of CUDA kernels execution through hardware-aware fine-tuning mechanisms. Specifically, we examine thread and block coarsening, 2 different CUDA kernel transformations that merge together kernel's threads and blocks respectively in order to deal with the problems associated with extensive fine-grained parallelism. Block coarsening refers to the transformation that merges together the workload of 2 or more threads from neighbouring blocks in contrast with thread coarsening that merges together neighbouring threads within the same block. Even though they both reduce the total number of each kernel's threads and they increase each thread's workload, they distribute threads in a different way across GPU's resources and thereupon affect kernel's execution at runtime differently. Figure 21 presents a simplified architectural view of both coarsening optimizations, illustrating how software transformations affect GPU's hardware resources.

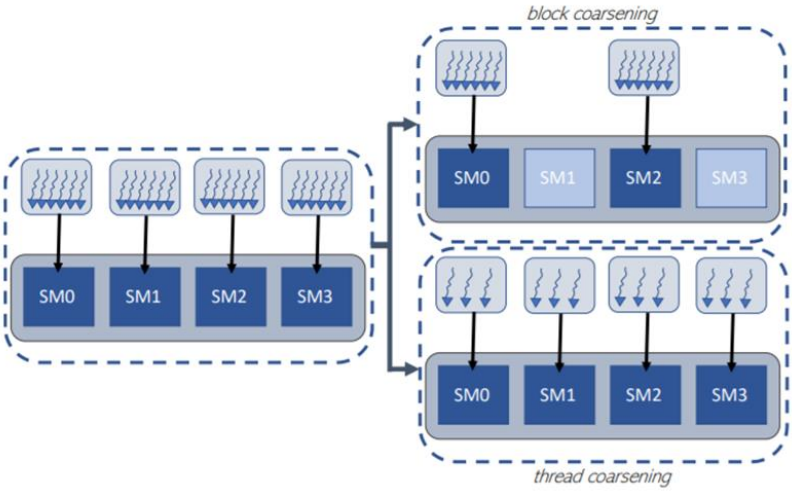


Figure 21: Block and thread coarsening transformations from hardware's perspective

GPUs are formed by multiple units named SMs (Streaming Multiprocessors). Each SM can execute many threads concurrently which are grouped into thread groups called warps containing 32 threads each. A warp is considered active from the time its threads begin executing to the time when all threads in the warp have exited from the kernel. From an architectural perspective, when the kernel is launched, GPUs map blocks to SMs (multiprocessors) and threads grouped in warps to CUDA cores. Therefore, as depicted in Figure 21, reducing blocks per kernel with block coarsening, reduces the number of occupied SMs, while thread coarsening reduces the number of threads per block and therefore the number of warps scheduled by each SM, while the number of SMs occupied by the kernel remains the same. However, even though both transformations attempt to improve parallel applications' efficiency, their impact seems to highly depend on exogenous factors such as device architectures, type of kernels, input size and data types. Hence, coarsening efficiently

kernels in both thread and block level remains a non-trivial task even for expert programmers in many cases. For that reason, accurate automatic optimization heuristics are necessary for dealing with the complexity and diversity of modern hardware and software as described in subsection 4.3.2.

4.3.1.2 Code optimizations for fine-tuned FPGA accelerated applications

On the other hand, the High-Level Synthesis (HLS) FPGA design flow aims to facilitate developers' effort to design optimal acceleration solutions for any algorithmic problem. In the HLS context accelerators are designed by firstly transforming the C/C++ algorithm to an equivalent FPGA friendly[2] code and then applying optimization directives at the code regions[3] that have been identified as computationally intensive. The word-length optimization directives are instructions to the HLS compiler regarding the implementation of those algorithmic tasks on the FPGA. To minimize the algorithm's latency and maximize its throughput the computationally intensive tasks should be executed either concurrently or in a pipeline mode, hence HLS tools provide directives that specify the kernel's execution on the hardware platform. Specifically, for executing loop iterations in a pipeline manner the designer applies the corresponding directive inside the loop body and sets the pipeline's initiation interval. Similarly, for executing multiple loop iterations in parallel, the designer specifies the desired level of parallelism through the corresponding HLS directive. Besides the word-length optimization instructions that aim to improve the algorithm's performance, HLS tools provide similar directives for managing the FPGAs local memory blocks and the interfaces that FPGAs programmable logic region exposes to the processing system's side. The deployment of approximate computing techniques in the HLS context reduces the FPGA's resource utilization and power consumption, allowing the developer to apply further performance-oriented HLS optimization directives and increase the level of parallelism and the pipeline factor, improving the algorithm's performance. Techniques that prune the least significant bits of the arithmetic operands or use pre-calculated values for complex arithmetic tasks such as logarithmic and sinusoid functions can be easily applied at FPGAs and lead to designs similar to the original ones in terms of quality but with reduced resource utilization and minimized power consumption.

4.3.2 Automatic Optimization Heuristics of GPU and FPGA accelerated kernels

Even though employing fine-tune implementations on both GPU and FPGA accelerators enables performance and power efficient executions, studying the mentioned optimization problems and building hand-written heuristics constitutes a time consuming and demanding task, that often leads to suboptimal solutions. Selecting and tuning manually appropriate features is a difficult task even for human experts, mainly because of the large design decisions space and its variation through different hardware devices. This process consists of different

[2] synthesizable code that allows the application of HLS optimization instructions
[3] for loops or functions

stages such as profiling the application in order to identify the computationally intensive parts (e.g., loops) and the design space exploration (DSE) for maximizing the algorithmic performance and minimizing the platform's resource utilization and power consumption. Hence, accurate automatic optimization heuristics are necessary for dealing with the complexity and diversity of modern hardware and software in order to avoid time loss and inaccurate code transformations.

The SERRANO project aims to fill the technology gaps by introducing a novel mechanism that automates the optimization process by building automatic machine learning heuristics able to predict optimal decisions based on representative features of unseen programs. It is within the goals of this project to develop a tool that automatically applies hardware specific optimizations for GPUs and FPGAs, to the computationally complex parts of an application. This mechanism will operate in two distinct phases:

- Identifying the computationally intensive application tasks: As a first step the tool will profile the application source code and detect the program regions that are mostly responsible for the application's performance degradation.
- Applying automatically the optimizations: Then, the mechanism will select the most suitable optimization technique and apply the method that maximizes the performance. For example, in the FPGA context, high level synthesis (HLS) directives will be applied, transforming the sequential code execution to pipelined or parallel. The tool will not only transform the original code but also define the appropriate level of optimization, such as the loop unrolling factor, with respect to the target platform's specifications. On the other hand, for the GPU case, coarsening optimizations will be automatically applied to CUDA kernels in order to meet the desired requirements. Given new kernels and target hardware devices, optimal block and thread coarsening transformations will be applied with the predicted coarsening factors in terms of minimizing tasks' latency and power consumption. In both cases, the mechanism will be based on a machine learning (ML) model that extracts features from the raw source code and predicts the optimizations.

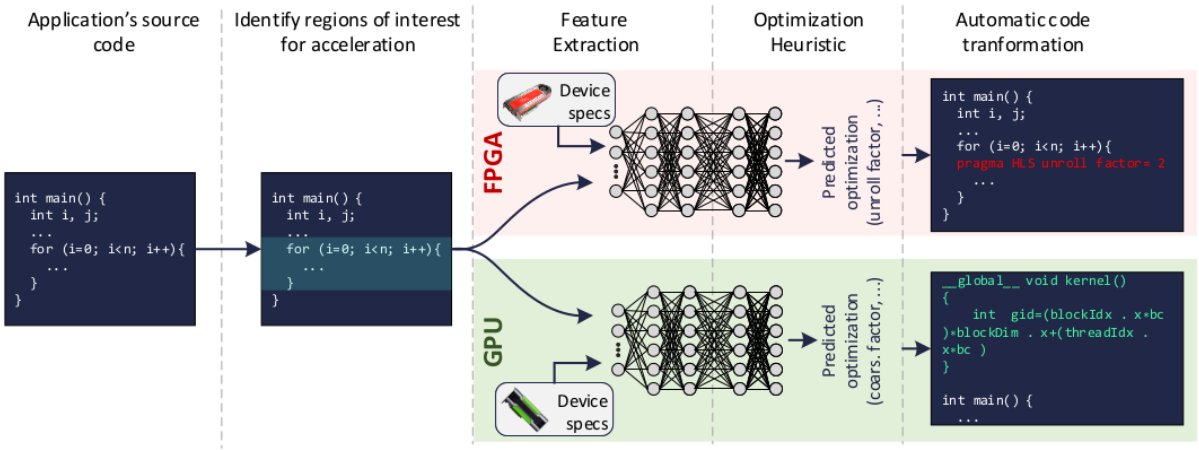


Figure 22: SERRANO's automatic optimization process of HW accelerated kernels

This automated optimization flow aims to ease developers from performing time-consuming optimizations, speeding-up the acceleration workflow and hence minimizing the application's time-to-market (TTM) period. Figure 22 conceptualizes the proposed scheme.

4.3.3 Dynamic Memory Management of FPGA accelerated kernels

This section discusses the incorporation of dynamic memory management during High-Level-Synthesis (HLS) for effective resource utilization in many-accelerator architectures targeting FPGA devices. Today, the main limiting factor of scaling the number of accelerators is the starvation of the available on-chip memory. For many-accelerator architectures, this leads in severe inefficiencies, i.e. memory-induced resource under-utilization of the rest of the FPGA's resources. Recognizing that static memory allocation - the de-facto mechanism supported by modern design techniques and synthesis tools - forms the main source of "resource under-utilization" problems, we introduce the DMM4FPGA framework that extends conventional HLS with dynamic memory allocation/deallocation mechanisms to be incorporated during many-accelerator synthesis.

The proposed DMM4FPGA framework enables each accelerator to dynamically adapt its allocated memory according to the runtime memory requirements, thus maximizing the overall accelerator count through effective sharing of FPGA's memories resources. The proposed framework will be integrated with the industrial strength Xilinx Vivado-HLS tool in order to improve productivity (e.g., by reducing design-time) for HW accelerators development. Preliminary results highlight that significant increase in FPGA's accelerators density is feasible in exchange for affordable overheads in terms of delay and resource count.

4.3.3.1 DMM4FPGA for supporting Dynamic Memory Management for HLS at Many-Accelerator Systems

Figure 23 shows an overview of the proposed DMM-HLS design and verification flow. The flow is based on Xilinx Vivado-HLS, a state-of-art and industrial strength HLS tool. The DMM-HLS extension works explicitly to the high-level source code of the application, thus it keeps minimum implementation overhead to the designers. A source-to-source code modification stage is the step where the original code is transformed from statically allocated to dynamically allocated using specific function calls from the proposed DMM-HLS API. For the targeted many-accelerator systems, the static-to dynamic-allocation code transformation is performed on data structures found within the global scope of the application. The transformed code is augmented by the DMM-HLS function calls and it is synthesized into RTL implementation through the back-end of Vivado HLS tool. DMM4FPGA framework extends the architectural template originally supported in Vivado-HLS by supporting emerging many-accelerator systems and allowing the on-chip BRAM to by dynamically allocated among accelerators at runtime.

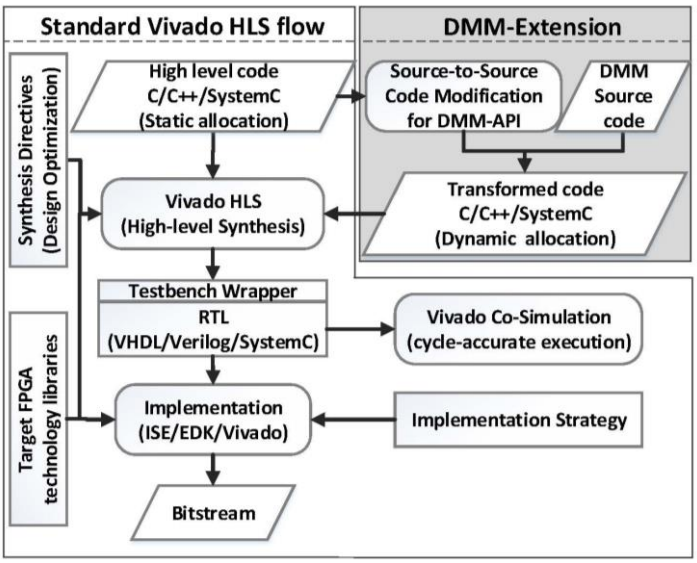


Figure 23: Proposed extension on Vivado HLS ow to support dynamic memory management for many-accelerators FPGA-based systems.

The proposed architecture, Figure 24, is divided in two subsystems: the processor subsystem and the accelerators subsystem. The processor subsystem typically consists of soft or hard IPs, such as the processor (e.g., ARM A9 in Zynq), executing three main code segments: the application control flow; the computationally non-intensive code kernels; and the code which shall be executed on CPU due to the need of high accuracy or high FPGA implementation cost (e.g., floating point division). The accelerators subsystem holds the computationally intensive kernels of the application. The accelerators are modelled in high-level language and synthesized using Vivado-HLS. The on-chip BRAMs are considered as a shared-memory communication link between processor and accelerators datapaths. DMM4FPGA framework supports the description of accelerators with both statically and dynamically allocated data stored in BRAMs.

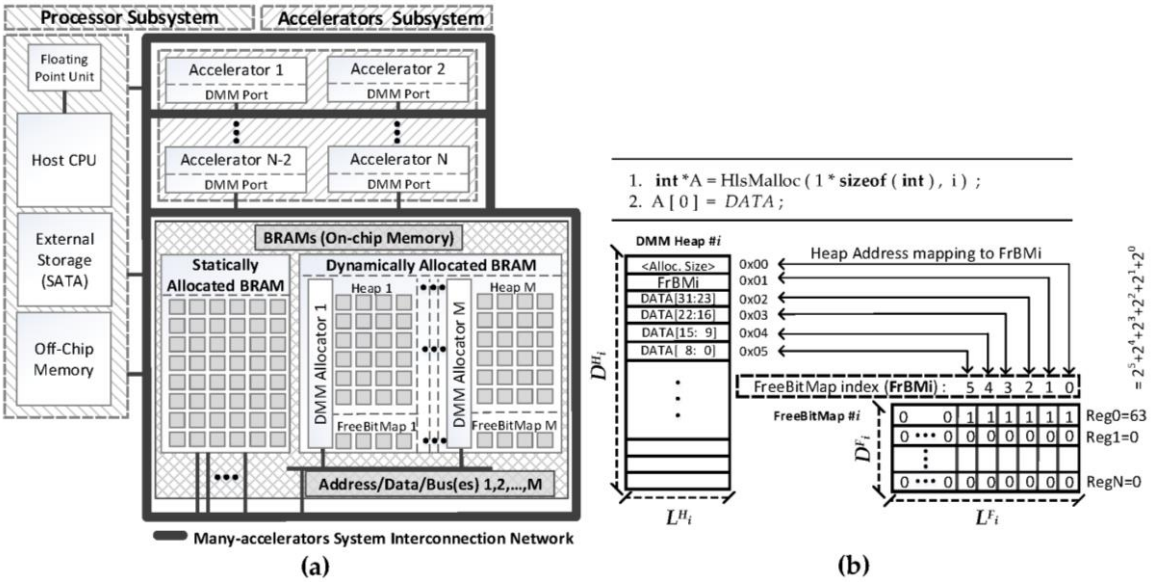


Figure 24 (a) Proposed architectural template for memory efficient many-accelerator FPGA-based systems. (b) Free-list organization, using a bit-map array

A key performance factor of the system is the ability to feed accelerators with data so that no processing stalling occurs due to memory read/write latency. Moving from static to dynamic allocation using an aggressive approach could lead to the allocation of all BRAMs under the same memory module so that all allocation/deallocation requests occur on this unique module. However, bigger memory banks mean wider wire buses that impact design closure (clock speed). In addition, throughput is limited by the BRAM ports (single or dual port) configuration. In the case of hundreds of accelerators parallel processing capability becomes highly memory constrained.

Similar to multi-threaded dynamic memory management [18][19] DMM-HLS supports fully parallel memory access paths, by grouping BRAM modules into unique memory banks, named heaps (see Figure 24). Every heap instantiation has its own DM allocator. The composition of heaps is described in a C header configuration file, enabling new heap instantiations to be easily described. Every heap-*i* is highly parameterizable on a number of design options, most important of which are (i) the heap depth D_i^H (the total number of unique addresses), (ii) the heap word length L_i^H (the number of bytes of every single word of heap), (iii) the allocation alignment A_i (the minimum number of bytes per allocation so that every new allocation starts from a unique address) and (iv) the meta-data header size H_i (the number of bytes reserved on first address(es) of every allocation to store meta-data related to the allocation, e.g. allocation length). In a similar manner we define the free depth D_i^F and free word length L_i^F for the FreeBitMap memory.

4.3.4 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 run-time adaption among approximate kernels in order 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 4.3.1-4.3.3, 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.

4.3.5 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 has also to 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 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. Depending on the goals of the user application, different techniques can be used. For example, if the signal analysis is relatively 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.

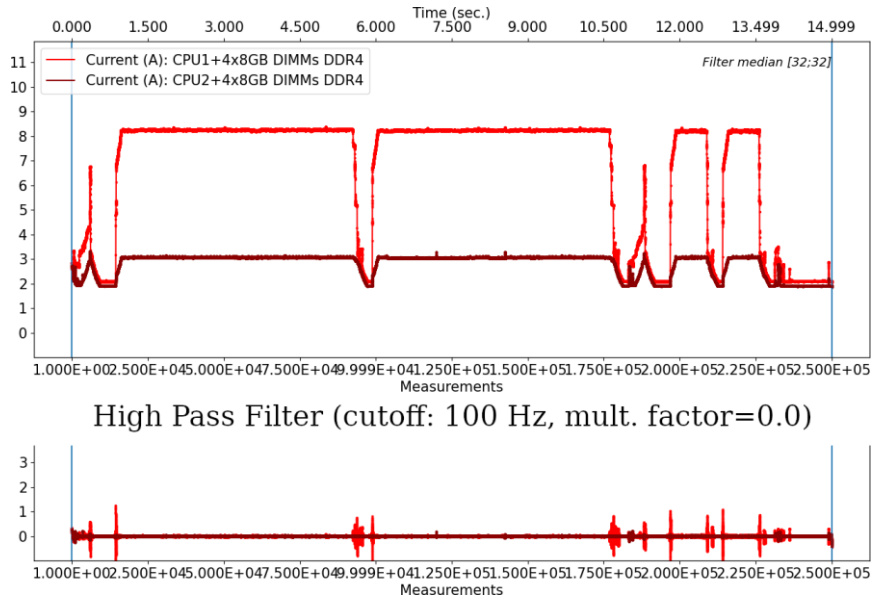


Figure 25: Signal from sensors for measuring the power consumption of two processors (measurement with 16.6 KHz) and the results of applying on it "high pass" FFT filter.

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 25 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 less data for the transmission of the results of filtering.

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)[4]. 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 deformation of the signal.
- When using the ASCII format, the data must be converted to the format supported by the hardware anyway. 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 and application 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 generate the report on the differences.

4.3.6 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 supply chain and build the implementation and reaction plans for a particular execution scenario. The **implementation component** is in charge of executing the generated supply chain and applying the provided implementation plan. Also, the monitoring system, capable of gathering data from all the resources or

[4] 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.

applications, must be setup and ensure that all the monitoring data is collected, processed, and made available for other components to consume it. The **diagnosis component** will consume the monitoring data and analyse and classify the collected monitoring events to identify possible 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 from 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 diagnosis component will be automatically configured by the orchestration mechanisms 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 (BSI) model. The beliefs represent the metrics (i.e., type, value, minimum and maximum, frequency or other types of measurable conditions) that describe a component specification, the desires represent the targeted quality assurance policies based on the component 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 from the service assurance and remediation system.

Figure 26 shows the main components of the service assurance and remediation mechanisms and Table 15 provides their description.

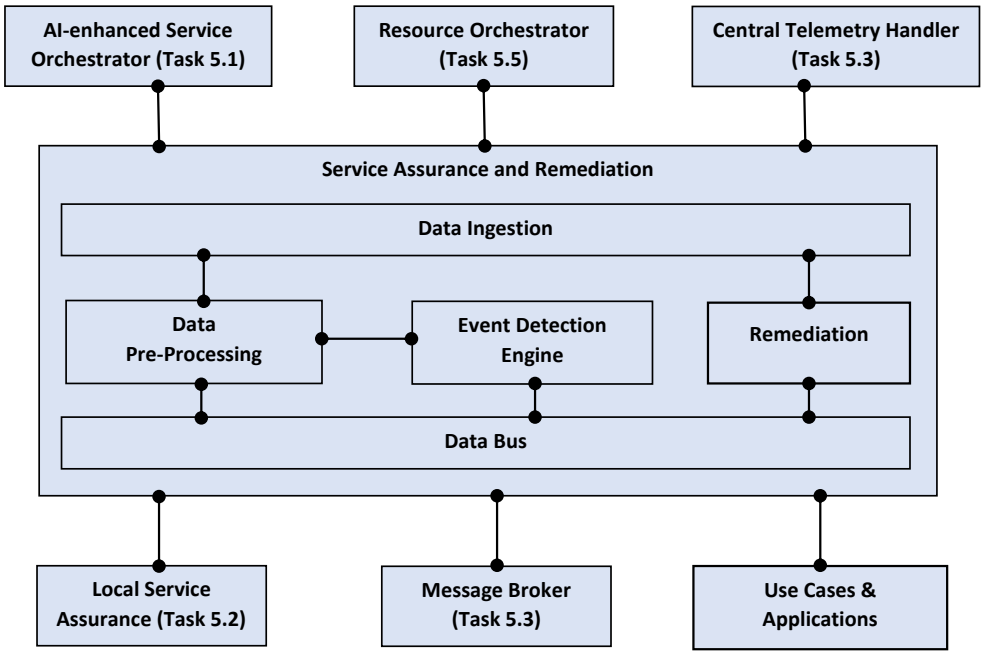


Figure 26: Service assurance and remediation system core components and dependencies

Table 15: 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 in 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 around supervised and unsupervised methods). One of the optimization methodologies is geared towards transprecise adaptation, meaning

	that a computationally optimized version of the detection models can be deployed on local service assurance instances. Once anomalies are detected these are reported using the SERRANO Message 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 and 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 reinstate an execution plan to initial or normal execution parameters. This component will follow a BSI 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.

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 proactively or reactively adjust resources and deployed applications. An autonomous, scalable and data-driven network and cloud telemetry framework will collect and analyse the required telemetry data across the distributed heterogenous (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 will leverage four 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 both by the typical resources and by the SERRANO-enhanced hardware resources. 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 in SERRANO platform cloud-native applications and short-lived (serverless) services will be automatically monitored.

SERRANO targets a hierarchical telemetry infrastructure to intelligently and dynamically monitor all the above categories of resources, 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 one coordinates the operation of a specific set of Monitoring and Logging Probes. Additionally, they pre-process, aggregate and disseminate the telemetry data towards the Central Telemetry Handler.
- *Monitoring and Logging Probes*: a set of probes (each one responsible for a specific resource type) that collect the actual telemetry data. Some probes will be out of the shelf, while others will be implemented in the context of SERRANO.

Figure 27 shows the key building blocks of the cloud and network telemetry infrastructure, their main components and the interactions with other SERRANO components.

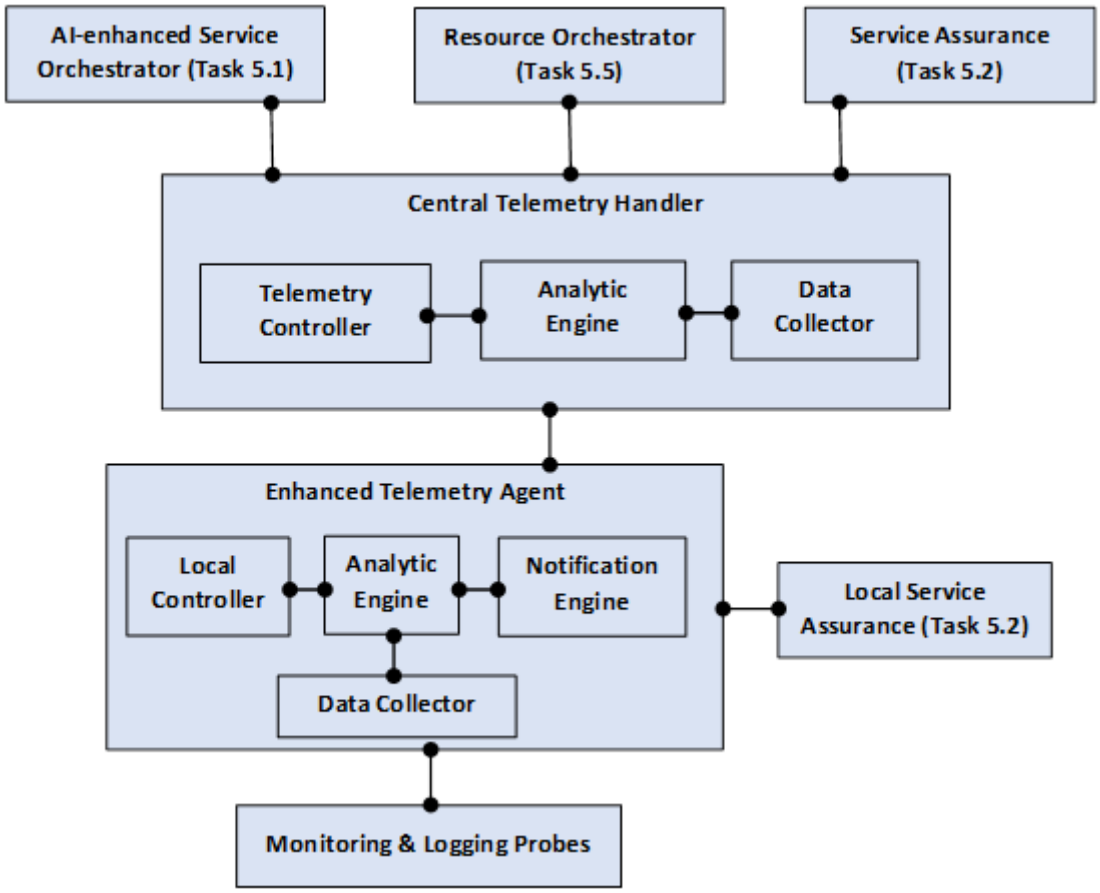


Figure 27: Cloud and network telemetry core components and dependencies

The following tables provide an initial description of core components along with the role of each one. The detailed design and implementation of the cloud and network telemetry within SERRANO will be the primary focus of Task 5.3, and the respective technical presentation will be provided initially as part of deliverable D5.3 “Resource orchestration, telemetry and lightweight virtualization mechanisms” (M15) and finalized in D5.4 “Intelligent service and resource orchestration mechanisms” (M31).

Table 16: 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 will maintain a global view of the current state of both the infrastructure and the deployed applications. It will coordinate 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 AI-enhanced Service Orchestrator and Resource Orchestrator. Hence, it will contribute to the cognitive and resource orchestration and deployment of UC applications in the SERRANO platform.

Component ID	WP5T3AE
Name	Analytic Engine
High level description	It will provide the analytic logic in the telemetry framework. It will include the appropriate algorithms that will rely on certain metrics 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
UCs	It will analyze 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 will be available both in the Central Telemetry Handler and the Enhance Telemetry Agents. In the first case, it collects the information from the Enhanced Telemetry Agents, while in the second by the various Monitoring and Logging Probes.
Collaborators	- Monitoring and Logging Probes - Enhanced Telemetry Agent - Analytic Engine
UCs	It will feed the Analytic Engine with the required data.

Component ID	WP5T3ETA
Name	Enhanced Telemetry Agent
High level description	It will coordinate the operation of a specific set of Monitoring and Logging Probes that are responsible for monitoring the resources and services to their administration domain. It will adjust their operation based on (i) the configuration commands from the Central Telemetry Handler and (ii) feedback from its Analytic Engine.
Collaborators	- Monitoring and Logging 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 will coordinate the operation of the Monitoring and Logging 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 and Logging 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 will enable 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 and Logging 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 project's 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 provide the interconnection between 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 in the overall platform the Data Broker component to support both message brokering and distributed streaming capabilities through the abstraction and integration of the appropriate message infrastructure. 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 is no intermediate components.

In addition, SERRANO's distributed streaming platform will allow to publish and subscribe to streams of records. This part of the messaging infrastructure will support high throughput and high-velocity data streams through a scalable, fault-tolerant communication-efficient framework. This approach provides the ability for asynchronous communication between SERRANO platform components as well as deployed applications.

The implementation of the Data Broker will be based on existing and well-known software platforms, with popular choices being Apache Kafka [20], Apache ActiveMQ [21], RabbitMQ [22], which support critical features like the loose coupling of components, increased scalability and security. Figure 28 shows the key building blocks and their interactions with other SERRANO components, while Table 17 provides an initial description for each component.

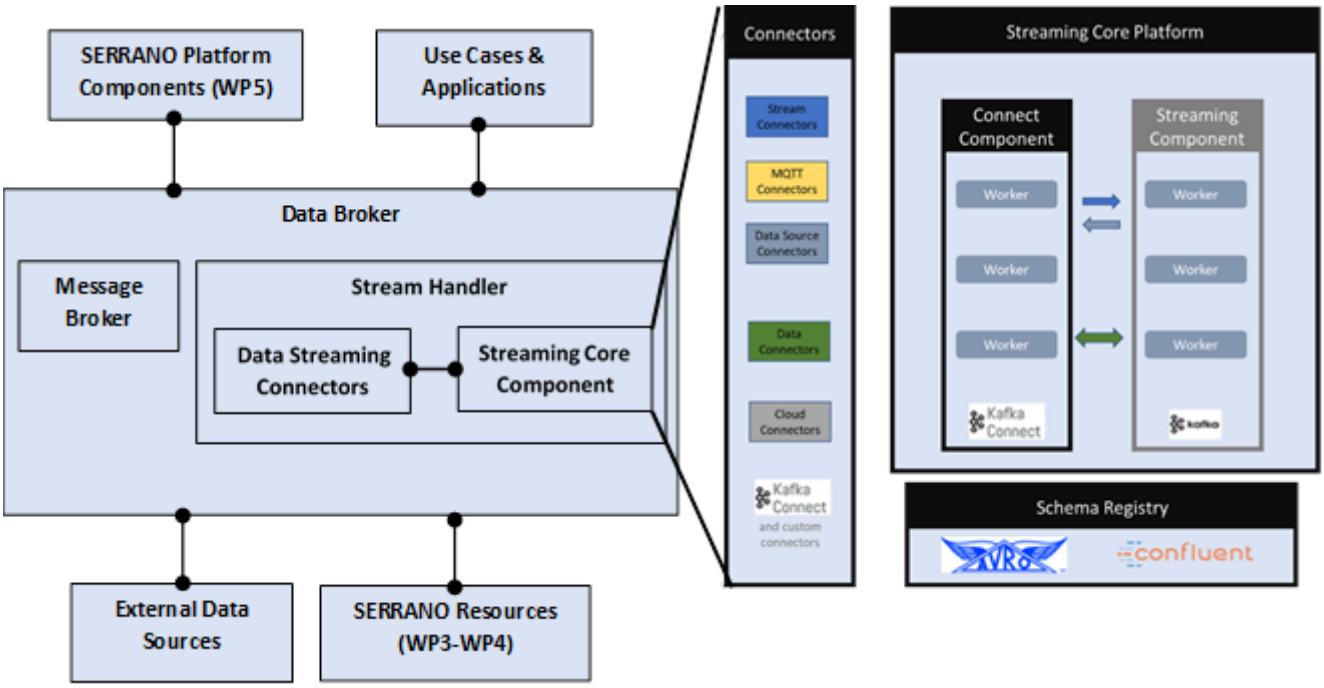


Figure 28: Data Broker core components and possible integrations with data sources and other infrastructure

Table 17: Description of data broker core components

Component ID	WP5T5MB
Name	Message Broker
High level description	It will facilitate asynchronous communication between the various components in the SERRANO platform. Moreover, it will provide similar functionality to the deployed applications. The component will leverage the publish and subscribe communication pattern to facilitate the loose coupling of the components while serving multiple senders and multiple receivers simultaneously. Message Broker will also support 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 will connect various external data sources that provide streamed data and make them available to the SERRANO platform, supporting the functionalities of the use cases.
Collaborators	- External data sources - SERRANO platform components - Deployed applications
UCs	It will handle incoming streams of data that need special handling or transformation before entering 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 will provide 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 29. In the selected architecture, 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-enhanced hardware (Section 3.1) and software (Section 3.2) resources. That exceptional unification of highly diverse resources provides the SERRANO platform the ability to cater for 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. A 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.

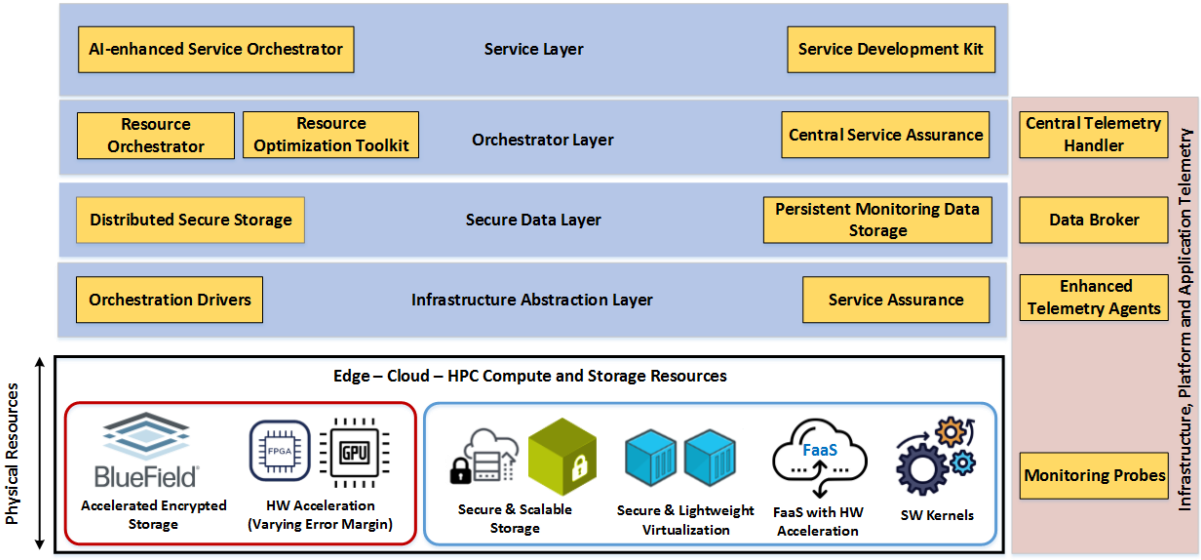


Figure 29: SERRANO high-level architecture

Across the SERRANO ecosystem resides the **Infrastructure, Platform and Application Telemetry stack** that tracks and logs 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, and is realized by the SERRANO hierarchical network and cloud telemetry mechanisms whose main components are the *Central Telemetry Handler*, the *Enhanced Telemetry Agents* and *Monitoring Probes*. The *Central Telemetry Handler,* which is the root in SERRANO's telemetry hierarchy, collects thorough 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 will facilitate the loose coupling of SERRANO platform services and abstract the management of applications input and output data through the provision of asynchronous data transfer and distributed data streaming functionalities. Furthermore, it will expose 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** that abstracts the peculiarities about 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 will be 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 the activation of self-driven adaptations for the deployed applications in each individual 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 edge, cloud and HPC platforms. The *Distributed Secure Storage* abstracts the required actions for each resource type, operating as a security access broker that guarantees and enforces privacy and security requirements on data. The *Persistent Monitoring Data Storage* allows the management of the historical monitoring data, necessary mainly by the service assurance and remediation system. The component needs to expose an interface through which store and query operations can be performed to tackle the historical monitoring data.

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 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 individual tasks constraints, while it also automatically coordinates the necessary supplemental actions (e.g., transfer 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, with special emphasis on energy consumption, robustness and latency. The *Central Service Assurance* component manages the runtime lifecycle of each application deployment across the SERRANO heterogeneous infrastructure. According to service specific needs, the infrastructure current state and notifications from the *Central Telemetry Handler*,

and the service assurance and remediation mechanisms at 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 analyses applications to automatically determine the most appropriate platform type for deployment and translates their high-level requirements into specific infrastructure related operational constraints and orchestration objectives. Moreover, the *SERRANO SDK* (Section 5) 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 requirements analysis and design (M01-M09) is presented in Figure 30. It includes all the components that will be developed by the project technical work packages (WP3-6), previously presented in Section 3. In addition, the diagram illustrates all the interactions and required information exchanges. According to the project implementation plan, the final version of the SERRANO architecture will be reported in deliverable D2.5 “Final version of SERRANO architecture” (M18), including all the required revisions after the first verification and evaluation of the initial technological developments.

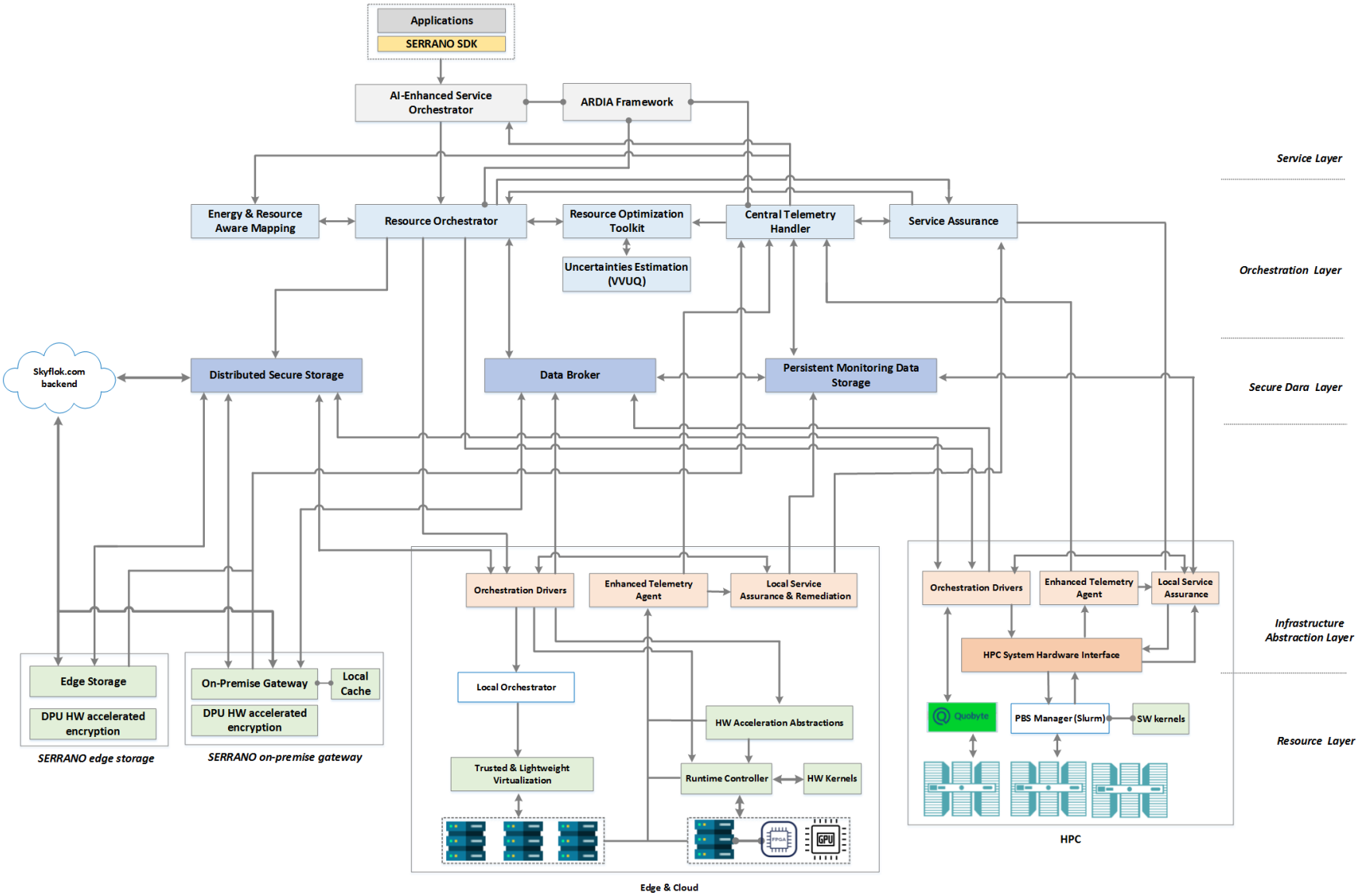


Figure 30: SERRANO detailed architecture

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 on 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).

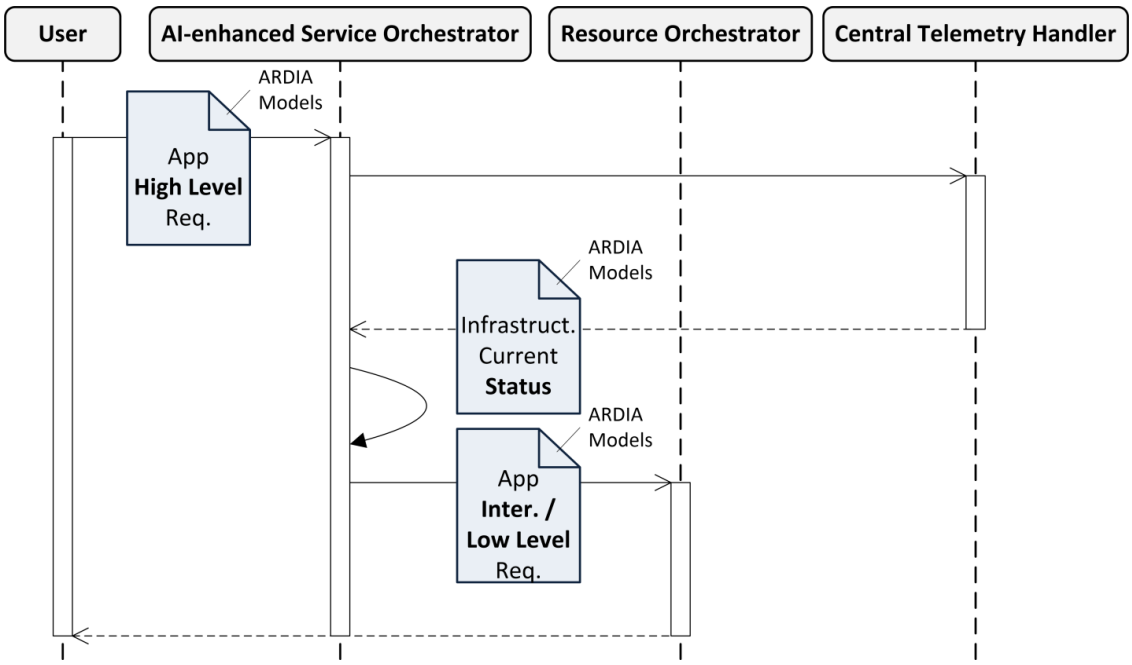


Figure 31: Interaction of AI-enhanced Service Orchestrator with the other components of the SERRANO platform

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

current status of the SERRANO infrastructure, in order to revise and update the suggested resource constraints (intermediate or low-level requirements). These constraints will be expressed based on the terminology specified in the ARDIA Framework and will be provided to the Resource Orchestrator, guiding its decisions.

5.2.2 Cognitive resource orchestration and transparent deployment

The next phase includes the initial allocation and orchestration of edge, cloud and HPC resources to efficiently fulfil the application requirements. The allocation of workload to the selected resources takes into account various criteria such as latency constraints, performance objectives, energy consumption and applications' security requirements.

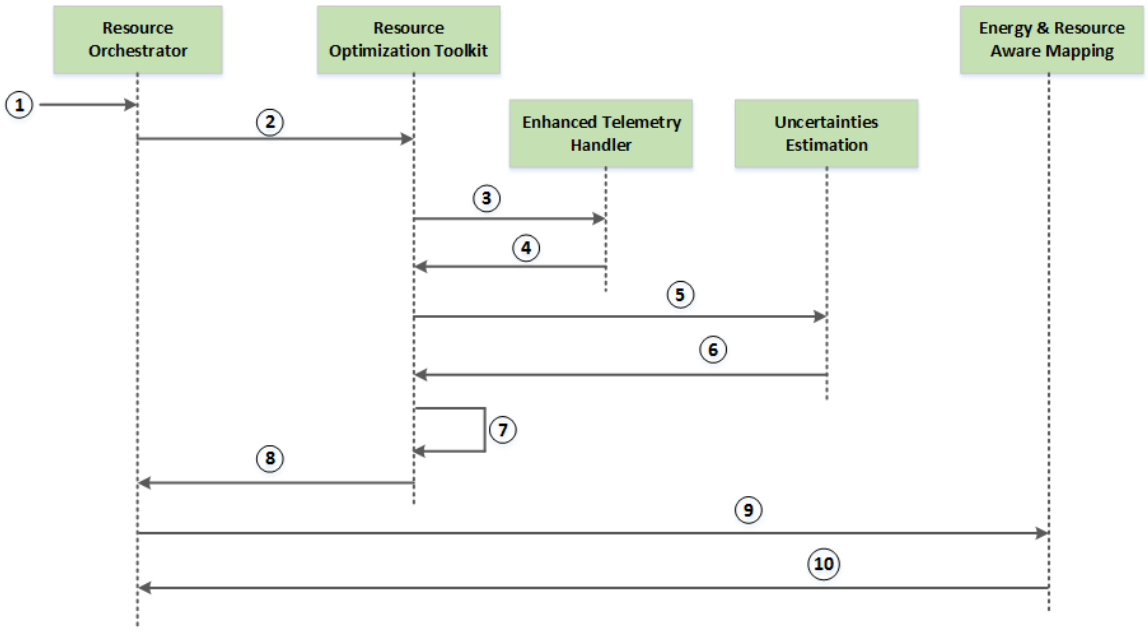


Figure 32: Cognitive resource orchestration operation within the SERRANO platform

Figure 32 showcases the involved components, and their interactions:

- The SERRANO Resource Orchestrator receives as input by the AI-enhanced Service Orchestrator the application description and a set of infrastructure specific deployment requirements and constraints.
- The Resource Optimization Toolkit (ROT) collects information about all the available resources through the Central Telemetry Handler.
- The ROT based on the available optimization algorithms decides on an initial application placement that both satisfies the user requirements and also improves the utilization of the available resources.
- The Uncertainties Estimation provides through the VVUQ framework useful 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 provides estimations about 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 33). This phase includes the following actions:

- The Distributed Secure Storage 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 setup the monitoring mechanisms for following up on the deployment status of applications over the selected edge, cloud and HPC resources.
- The Orchestration Drivers at the selected platforms receive the deployment instructions to perform the actual deployment with the selected runtime configurations.

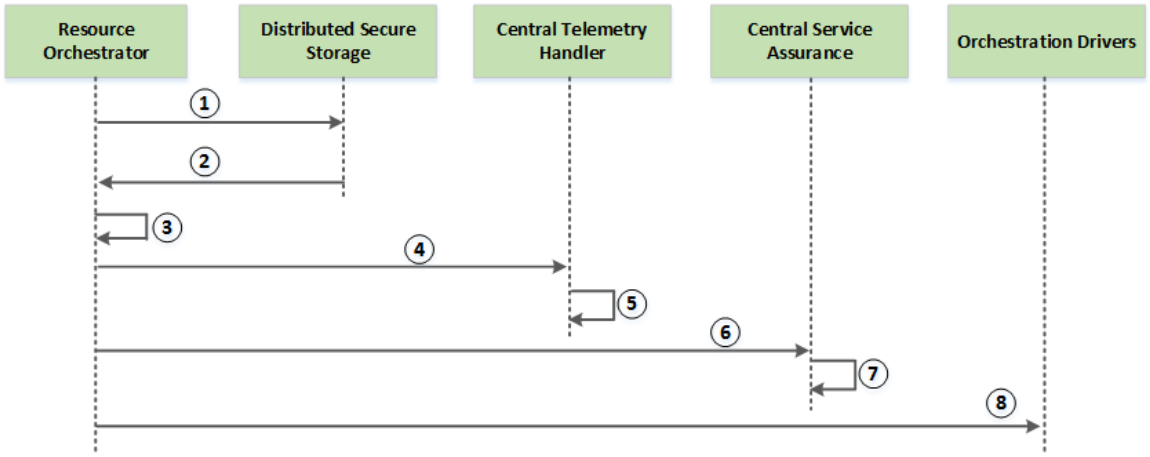


Figure 33: Transparent application deployment operation within the SERRANO platform

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

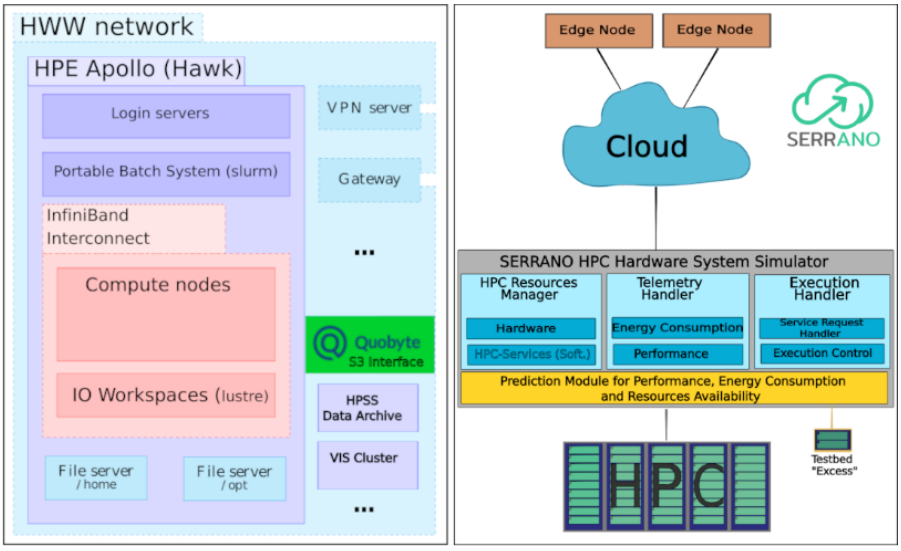
5.2.2.1 HPC Integration

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 34.a shows the internal HLRS network, so called HWW, that connects all HPC systems and its 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 not all telemetry data is exposed by an HPC platform. To this end, we will develop an additional layer between the SERRANO platform and the HPC systems, namely "HPC System Hardware Interface". This approach not only will enable the immediate integration of HLRS resources within SERRANO platform but also it makes the integration more universal and adaptable to various others HPC systems.

The HPC integration module (Figure 34.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., 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 34: (a) Internal HLRS's network HWW connects HPC resources at HLRS, (b) Main components of the HPC integration module within the SERRANO

Figure 35 depicts the interaction among the involved components for the deployment of an application in an HPC platform. The object file system "Quobyte" [23] will provide the required support for the input and output data transfer between the HPC system and the SERRANO platform. Quobyte supports 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 System Hardware Interface, the HPC system and forwards the corresponding information to the Central Telemetry Handler. During the deployment, the Orchestration Driver (Section 4.2.1) receives the deployment instructions, forwards them to HPC System Hardware Interface in order to map them to HPC-specific commands. Moreover, if necessary, the Orchestration Driver through the Secure Storage service and the Quobyte moves the required input data to the HPC. Next, the job is submitted to the Local Orchestrator (i.e., PBS Manager), while the Local Service Assurance is instructed by the Orchestration Driver to keep track of the submitted tasks through the HPC System Hardware Interface. Moreover, the Local Service Assurance informs the Orchestration Driver and the Central Service Assurance whenever the execution is finished. Finally, the Orchestration Driver is able to send asynchronous messages to any other component in the SERRANO platform through the Data Broker (Section 4.5).

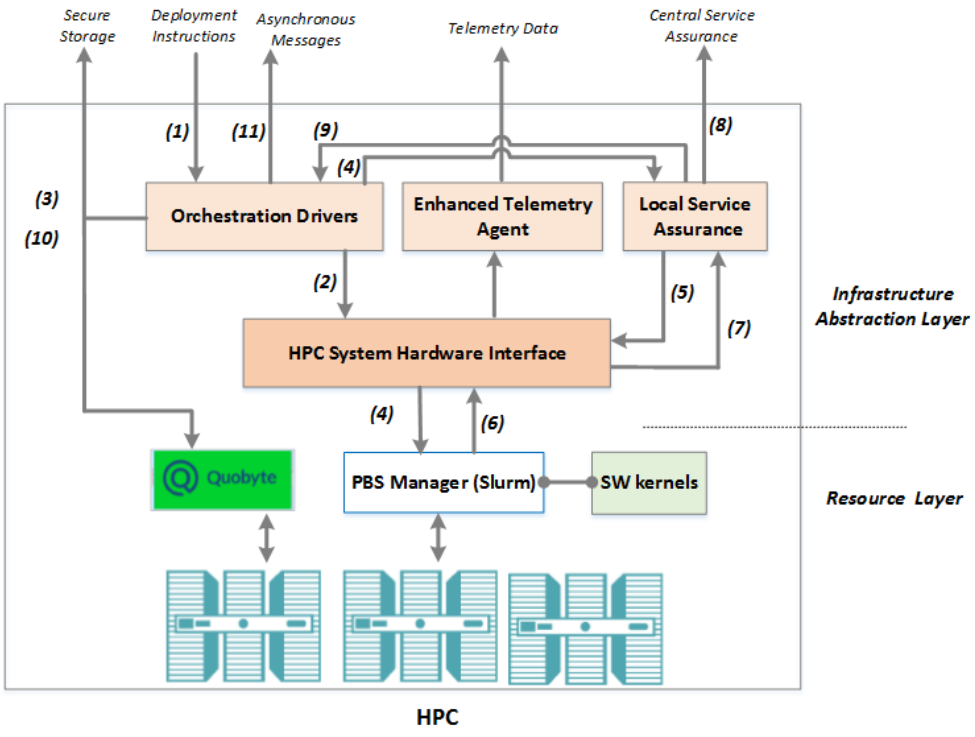


Figure 35: HPC integration and application deployment workflow

5.2.2.2 Edge and cloud integration

Figure 36 shows the interaction among the involved components that enable the transparent application deployment over edge and cloud platforms. Similar to the HPC integration, the Enhanced Telemetry Agent constantly monitors, through the appropriate Monitoring and Logging Probes, the underline resources and available applications and forwards the corresponding information to the Central Telemetry Handler. The actual deployment starts with the Orchestration Driver that translates the deployment instructions by the Resource Orchestrator to specific requirements for the Local Orchestrator. Moreover, the Orchestration Driver through the SERRANO Secure Storage service transfers the required input data.

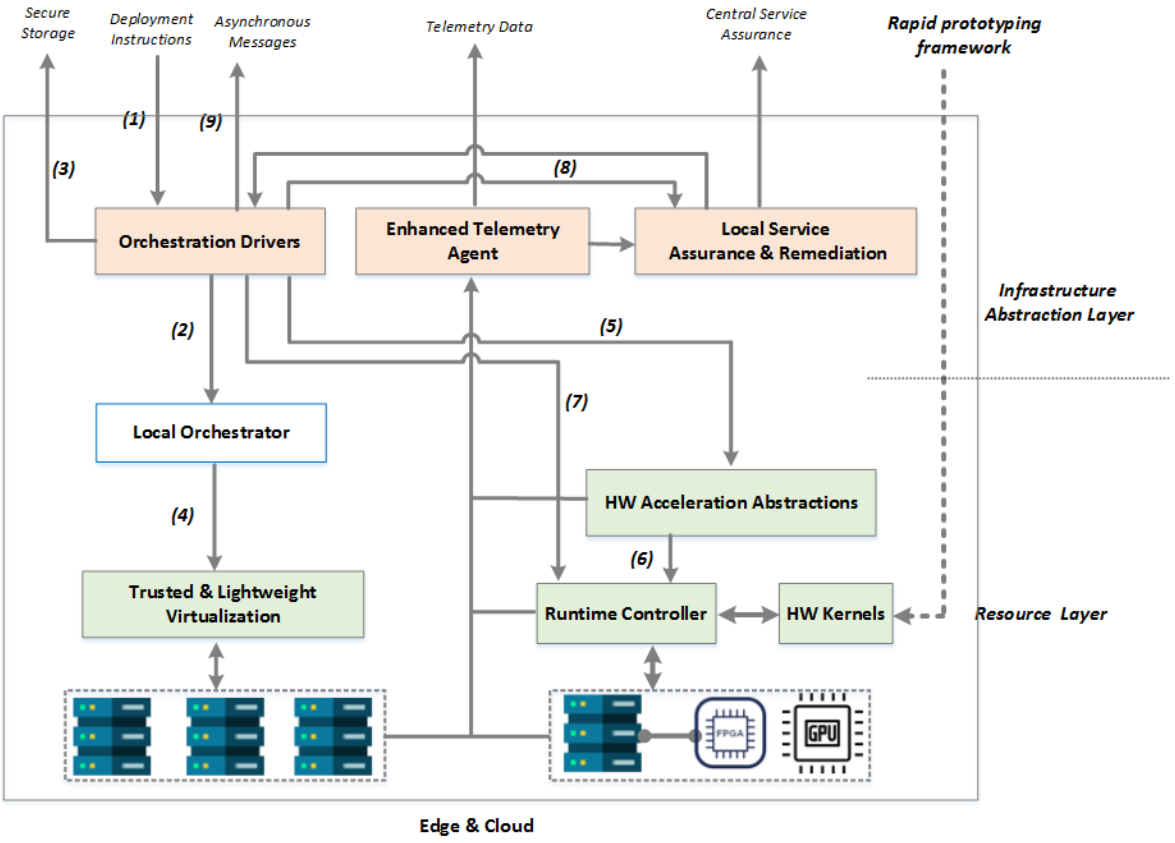


Figure 36: Edge and cloud integration and application deployment workflow

Next, the Local Orchestrator (e.g., Kubernetes) manages the submitted workload. Since SERRANO aims to provide support for more than just containers (e.g., VMs, unikernels, etc.), it will implement Trusted and Lightweight Virtualization mechanisms to provide the required glue layers that will enable the Local Orchestrator 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 stack by being compatible with the Open Container Initiative (OCI) interface. In SERRANO, we also add the required functionality to the container runtime in order to support vAccel, the hardware acceleration framework introduced in 3.2.3, which enables access to hardware acceleration resources without the need for direct hardware/device access.

We group the above functionality to the Local Orchestrator component, which provides: (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 at an abstract and disaggregated manner. To this end, when the Orchestration Driver interacts with the Runtime Controller for hardware approximate kernels, specifying the selected kernel and the requested quality-performance preferences (e.g., affordable error, energy). The hardware approximate 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 – quality requirements are met for each application. Finally, the Orchestration Driver is able to 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 Resource Orchestrator receives storage tasks and decides which storage service will serve them. To achieve this, the central orchestrator queries the storage services on the characteristics of the storage resources they manage (e.g. available data locations, the status of resources, cost, availability, latency, etc.). Also, the orchestrator's application deployment decisions are guided by this information. For example, information about the location of some data (after it has been stored using one of the available services) may influence the central orchestrator decisions regarding the assignment of computational workload to edge, cloud and HPC resources. This high-level view of storage orchestration is shown in Figure 37.

If the SERRANO Secure Storage Service is selected as the optimal solution, a second, more fine-grained level of orchestration is performed with the aim of matching the requirements of the application with the most suitable storage locations, erasure coding configuration and encryption scheme. This is done through the creation or selection of a storage policy, a configuration entity that acts as a recipe on how storage tasks are to be served. Deliverable

D2.2 went into detail explaining this concept and provided some examples. This process takes place on the On-premise storage gateway (Gateway from now on).

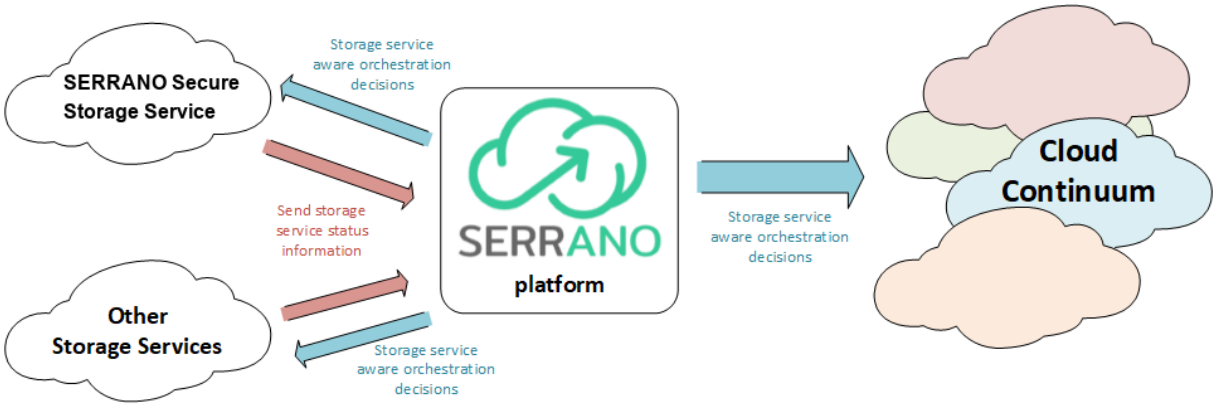


Figure 37: Storage service aware orchestration of storage tasks

SERRANO applications access the SERRANO Secure Storage through the Distributed Secure Storage interface (Section 6.2) when uploading and downloading files. Figure 38 showcases the upload process, which involves the application uploading the file to the Gateway. This is then cached, encrypted and several (four in the example on the figure) erasure-coded fragments are created. These are uploaded to a combination of cloud and edge locations (2 + 2 in the example). The fragment uploads are performed by the Gateway, after authenticating with the Skyflok.com backend. It is the backend that provides the upload links as pre-signed URLs, a technique described in Section 3.2.1.2.

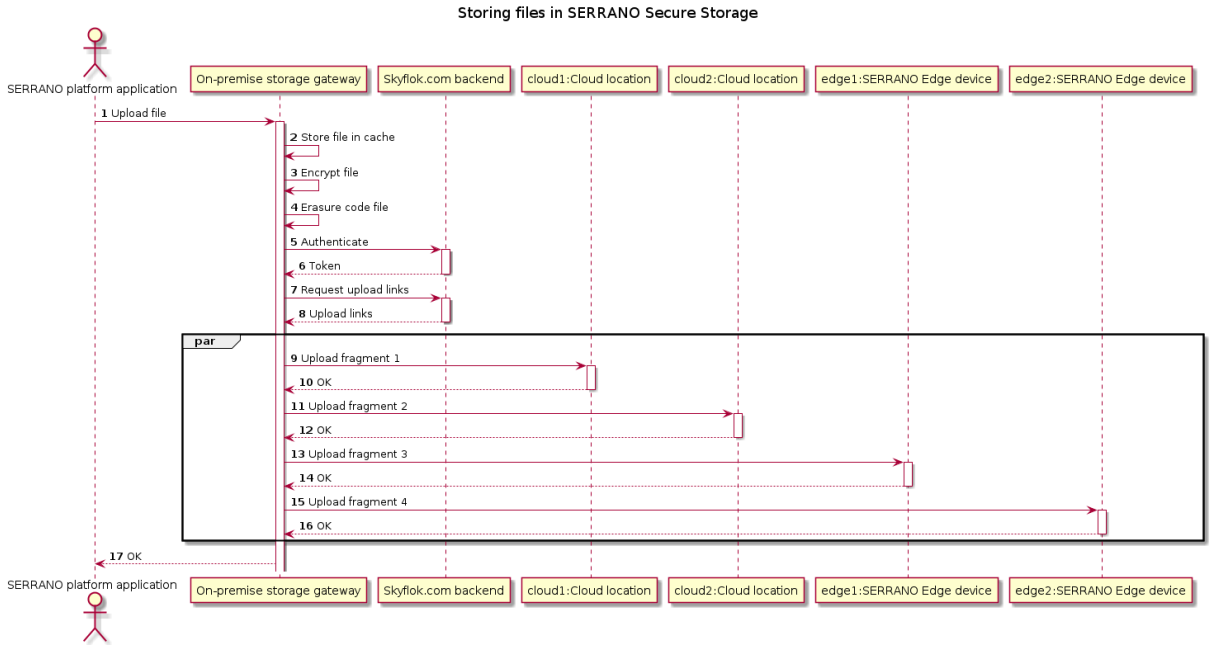


Figure 38: Sequence diagram showing file upload in SERRANO Secure Storage

Figure 39 showcases the download process. The SERRANO application requests the file using its filename, which acts as an identifier. 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 present in the cache, the gateway request 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.

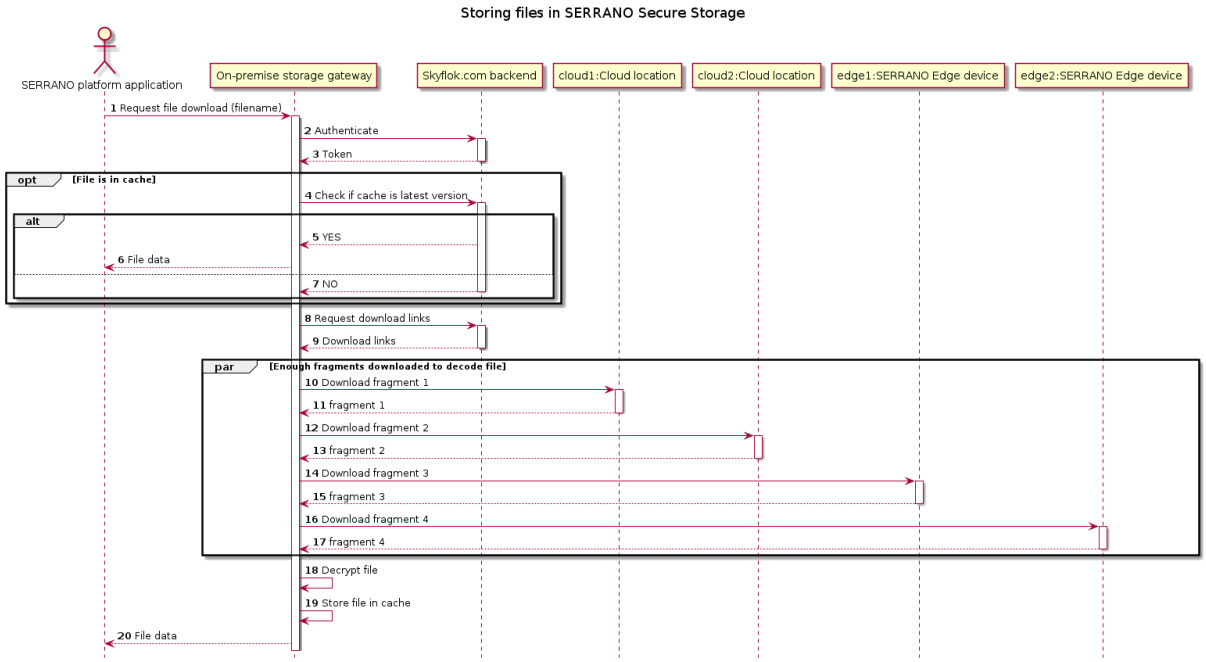


Figure 39: Sequence diagram showing file download from SERRANO Secure Storage

The actual movement of file data (both upload and download) is performed through the object storage interface (Section 6.2). This is in fact 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 40 presents the involved components, while this phase includes the following actions:

- The Service Assurance and Remediation (SAR) component get 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 anomalies detection, the SAR will issue and push alerts to the resource orchestration mechanisms through the Orchestration Driver for triggering 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 to apply it.
- The Orchestration Driver contacts the local orchestrator that, based on the current state of the existing resources and the remediation plan by the SAR, readjusts the initial deployment of the affected application.
- A special kind of reconfiguration may take place if the application involves hardware accelerated 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 accordingly the service assurance mechanisms at the orchestration layer.

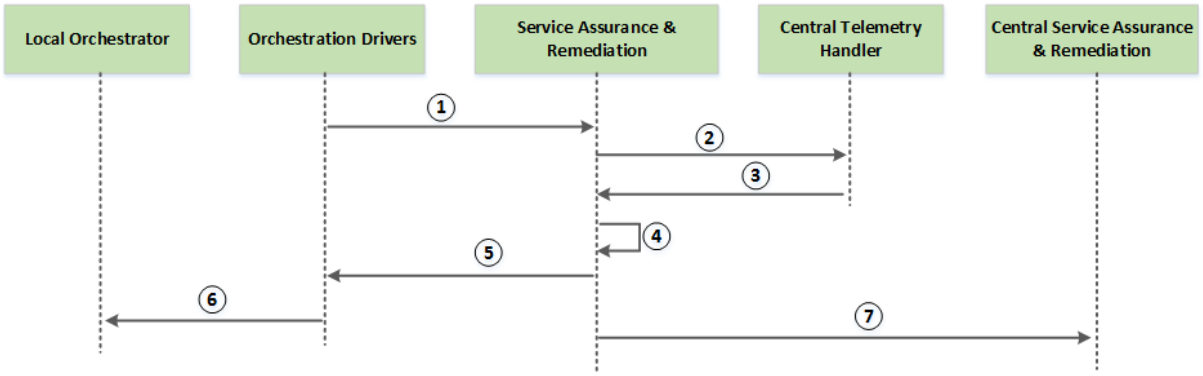


Figure 40: Service assurance and dynamic adjustments within the SERRANO platform

Additionally, the UCs that need to solve event detection problems can directly interact with the service assurance mechanisms. The application will communicate with the Event Detection Engine (EDE) component through the Message Broker or the Data Bus 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 presents a preliminary description of the interfaces required for the communication between the SERRANO components (Sections 3 and 4) that will be developed in technical work packages (WP 3-5).

6.1 Service and Resource Orchestration Interfaces

In this subsection, we focus on the main interfaces required for the communication between the core components at the service and orchestration layers. Specifically, the following tables provide the preliminary description of the exposed interfaces by the AI-enhanced Service Orchestrator, Resource Orchestrator, Resource Optimization Toolkit, Uncertainties Estimation and Energy and Resource Aware Mapping.

Table 18: 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 applications
Type of interface	REST
State	Synchronous / Asynchronous It depends on how this component will be used. The AI-enhanced service orchestration should synchronously translate and enrich the UC applications requirements and goals to the appropriate ones so that they can be used by the Resource Orchestrator. Nevertheless, the overall execution of each UC application may be asynchronous.
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 exchanged will be described in the deliverable D5.1.
Responsibilities	INNOV, ICCS

Table 19: Resource Orchestrator Interface

Interface ID	WP5T5RO-I
Description	This interface enables the AI-enhanced service orchestration to initiate the resource allocation and deployment of applications. 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 20: Orchestration Drivers Interface

Interface ID	WP5T5OD-I
Description	This interface enables the implementation of the hierarchical orchestration and application deployment within the SERRANO platform. It exposes the required methods to allow the Resource Orchestrator to send 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	AMQP
State	Asynchronous
Constraints	The description of deployment instructions should be generic and has to follow the ARDIA framework specifications.
Responsibilities	ICCS, NBFC

Table 21: Resource Optimization Toolkit Interface

Interface ID	WP5T2ROT-I
Description	The interface allows 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
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

Table 22: 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 Hardware System Interface
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, 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 23: Energy & Resource Aware Mapping Interface

Interface ID	WP5T4EMT-I
Description	SERRANO Energy & Resource Aware mapping concerns the efficient use of SERRANO platform components, in particularly the HPC systems.
Component providing the interface	SERRANO HPC Hardware System Interface
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 HPC Hardware System Interface.
Type of interface	Binary protocols
State	Asynchronous
Constraints	It will be investigated.
Responsibilities	HLRS, ICCS, INNOV

6.2 Distributed Secure Storage and Telemetry Interfaces

The Distributed Secure Storage interface (Table 24) will enable both the applications deployed to the SERRANO platform and the platform services (Section 4) to interact with the On-premise storage gateway. It follows the de-facto standard set by Amazon with its S3 object storage service, a design choice that allows for easy integration with existing applications designed to work with this API. It will provide support for 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 24: Distributed Secure Storage Interface

Interface ID:	WP3T2DSS-I
Description	This interface provides a way to upload and download files to/from the SERRANO Secure Storage service.
Component providing the interface	On-premise 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-premise storage gateway accesses the objects on the cloud storage locations through either proprietary interfaces, S3 or S3-compatible interfaces or the Swift API. Objects on SERRANO edge devices are also accessed through an S3-compatible interface, as provided by the MinIO S3 Gateway [24]. Table 25 summarizes the REST interfaces provided by each storage location.

Table 25: 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 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 26). This interface will be defined based on the specification of these services.

Table 26: 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	On-premise storage gateway (or Skyflok.com backend, it will be determined later on), SERRANO edge devices.
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 initial specifications for the provided interfaces by the main components of the SERRANO telemetry mechanisms and data broker services.

Table 27: 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. It will also expose methods that enable components to receive monitoring events for various situations.
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 28: Enhanced Telemetry Agent Interface

Interface ID	WP5T3ETA-I
Description	This interface is responsible for forwarding the monitoring information and the related events across the hierarchical telemetry infrastructure. It will also provide 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 29: 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 30: Message Broker Interface

Interface ID	WP5T5MB-I
Description	Message Broker Interface
Component providing the interface	The Message Broker will support communication using a protocol specialized for message exchange (e.g. AMQP) or directly through the TCP layer via compatible client libraries in other components.
Consumer components	SERRANO Platform components, use cases and applications, external data sources, SERRANO resources
Type of interface	Plain TCP, 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 31: Streaming Core Interface

Interface ID	WP5T5SC-I
Description	Streaming Core Interface
Component providing the interface	The Streaming Core Interface will support 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 32: Data Streaming Connector Interface

Interface ID	WP5T5DSC-I
Description	Data Streaming Connector Interfaces
Component providing the interface	The Data Streaming Connectors will enable 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 33: 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 will extend the Plug&Chip framework [25] for rapid prototyping, which in turn will be exposed as a service to the SERRANO platform. Users will have to provide their source code (C, C++) as well the requirements of their application (e.g., target Queries per second, Transactions per second, etc.) through the provided interface. Then, the Plug&Chip framework will perform a Design Space Exploration (DSE) to determine viable solutions in terms of different hardware accelerators and approximate kernels types in a seamless to the end-user manner. The DSE process will result in a library of viable (in terms of user constraints) kernels that trade-off between accuracy, performance and energy. These kernels can be leveraged by the orchestration mechanisms to satisfy QoS guarantees under different interference and/or stressing scenarios.

Table 34: Rapid Prototyping Interface

Interface ID	WP4T3RP-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	REST
State	Asynchronous
Constraints	Users leveraging this API should provide a C/C++ source code along with their application's requirements
Responsibilities	AUTH

Finally, the following tables provide the preliminary specifications for the required interfaces that will enable the smooth 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. This API will expose accelerators in case 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 Hardware System Interface
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 will expose workload execution functionality to the orchestrator and the end-users/applications. It will reflect 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 depends 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 will expose the developed APIs through a library.
Component providing the interface	SERRANO SDK
Consumer components	All Components that use the Serrano APIs will be able to do so through the SERRANO SDK
Type of interface	Python library
State	It will support the states that the SERRANO APIs can support.
Constraints	All constraints of SERRANO APIs will also apply to SDK. Also, the Python SDK will be usable though a Python App or script. 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 an initial high-level description of the separate deployments that SERRANO will setup for the evaluation of the UCs.

7.1 UC1: Secure Storage

The principal components of the Secure Storage use case (UC) are shown on Figure 41. The on-premise storage gateway and the SERRANO edge resources will run as containerized applications in the respective devices, deployed on the customer's premises. These will be managed by the SERRANO platform.

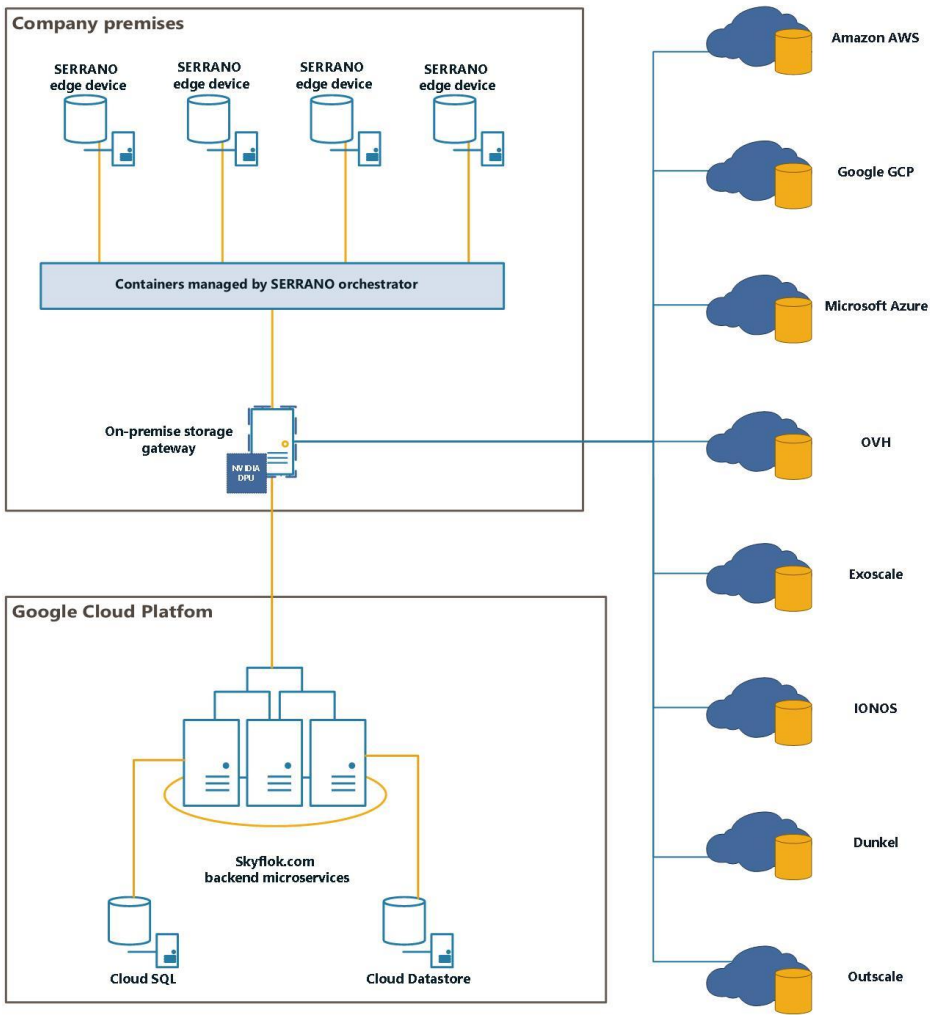


Figure 41: Secure Storage use case deployment overview

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). The UC 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 that will serve the storage tasks.

7.2 UC2: Fintech Analysis

The deployment of the FinTech use case (UC) for portfolio optimisation is displayed in Figure 42. It consists of deployment on, on-premise servers as well as cloud servers. The entire deployment will be managed by the SERRANO cloud orchestration and management components of the platform. The on-premise servers will execute functions that are not containerized, for example, obtain market data, create investment profiles, portfolio rebalancing and order operations. Multiple instances of microservices will be deployed through containers into SERRANO-enhanced nodes. Some of these microservices will use the available accelerators on the node that are deployed. For example, market analysis, forecasting and backtesting would benefit from the available acceleration on particular SERRANO-enhanced node. Other microservices such as for investment strategies, cannot benefit from any hardware acceleration and may be deployed on cloud nodes that do not have accelerators.

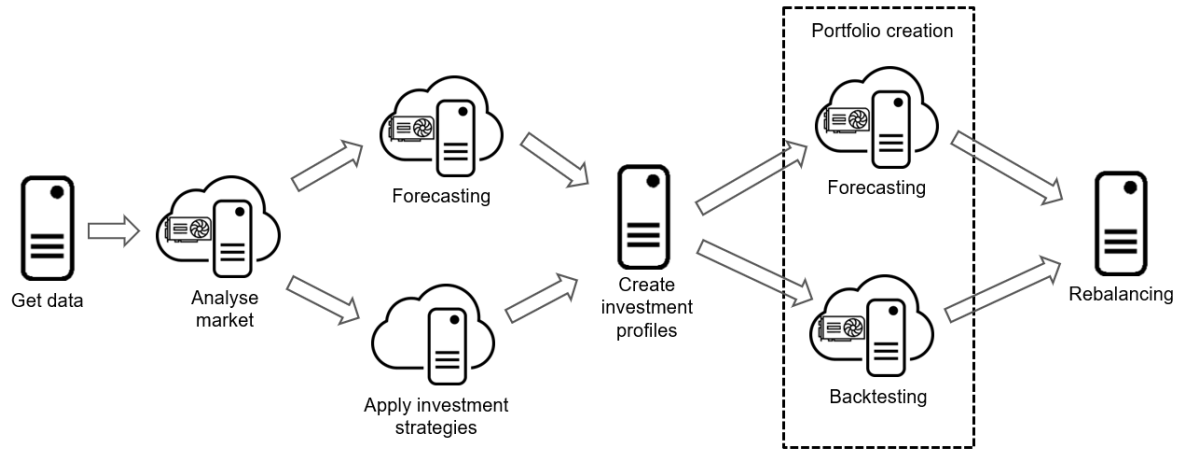


Figure 42: FinTech portfolio optimisation use case deployment overview

7.3 UC3: Anomaly Detection in Manufacturing Settings

The principal components of the Anomaly Detection in Manufacturing Settings use case (UC) are shown on Figure 43. When the machine starts machining, data are continuously/periodically generated. Data are being monitored by a mechanism running on the SmartBox attached to the machine, which is an edge device that 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 (Section 4.5).

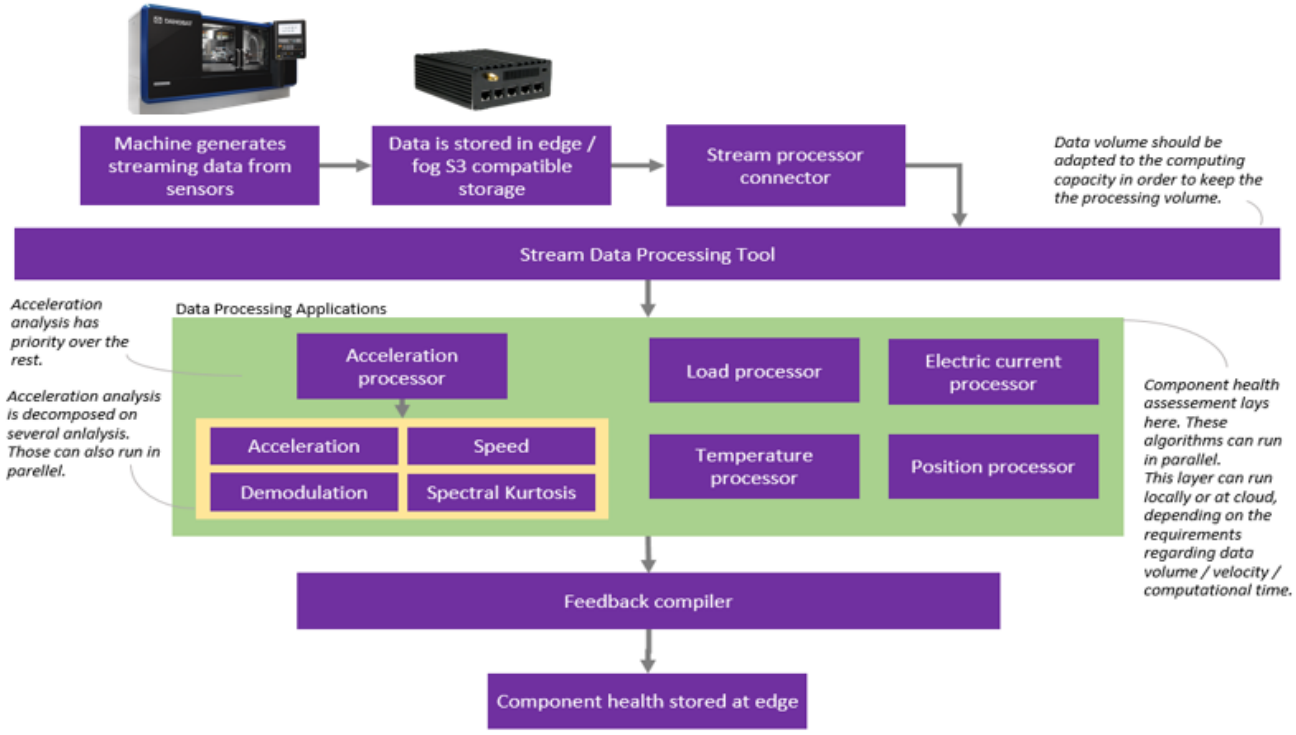


Figure 43: Anomaly Detection in Manufacturing Settings use case deployment overview

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. The processor service is where system intelligence resides. Data coming from accelerometers is sent to the Acceleration Processor, while data coming from Temperature sensors will be send to the Temperature Processor. As an example, the Acceleration Processor may use Fourier Transform to look for abnormal measurements over different frequencies; the Load Processor may use some anomaly detection algorithm to look for rapid fluctuations. Also, in this layer (Data Processing Application) SERRANO could decide the data accuracy to apply depending on the required latency, energy consumption or available computation resources. Any Data Processing Application can run at a SERRANO-enhanced resource (edge, cloud, HPC) depending on the client's requirements and the SERRANO orchestration decisions. In the end, the results of the analysis performed by the processing services are sent to a Feedback compiler layer and then stored locally for further use. This UC may utilize the secure storage layer of SERRANO, exhibited through UC1.

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 practice in CI is to integrate the changed code at least daily. The frequent integration helps the contributors to notice any arising errors and correct them instantly.

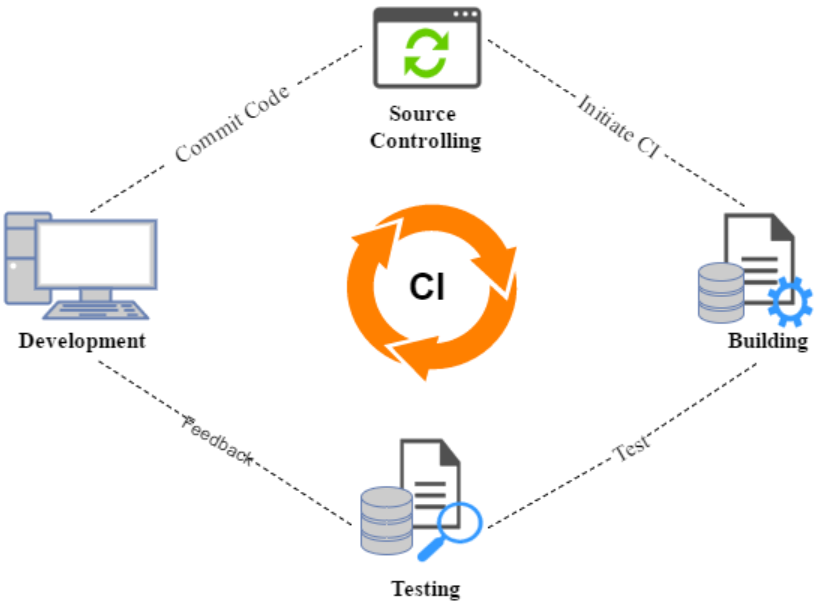


Figure 44: The Continuous integration lifecycle

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 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:

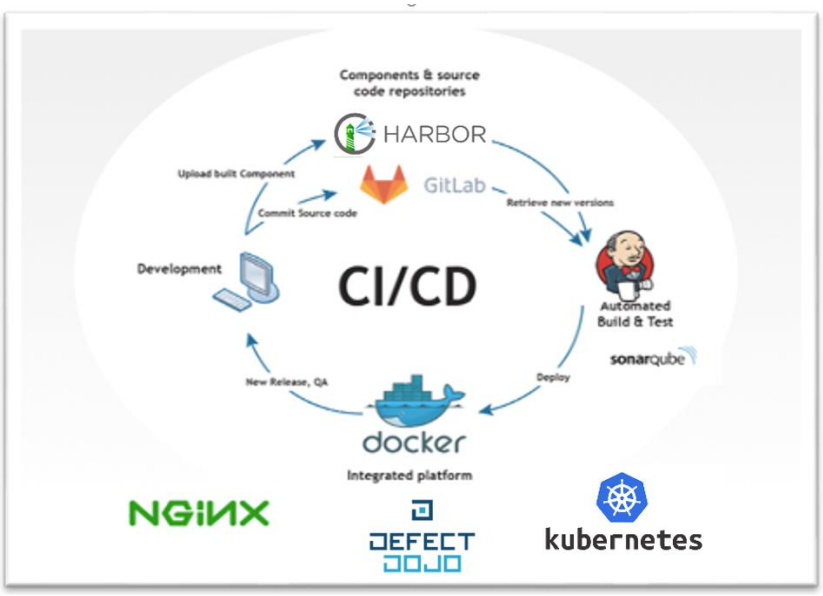


Figure 45: CI/CD tools in development workflow

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 SERRANO project is organized based on a set of well-defined and complementary phases (Figure 46) 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-M20, M20-M36), each of which will include a series of activities, which bridge the gap between requirement analysis, technology and innovation. The developed components and services will be continuously integrated, according to the integration plan (Section 8.1), with the defined interfaces and communication protocols as set in the SERRANO architecture specification. The project will conclude with the demonstration of the UCs (Phase 5) and the exploitation of the results (Phase 6).

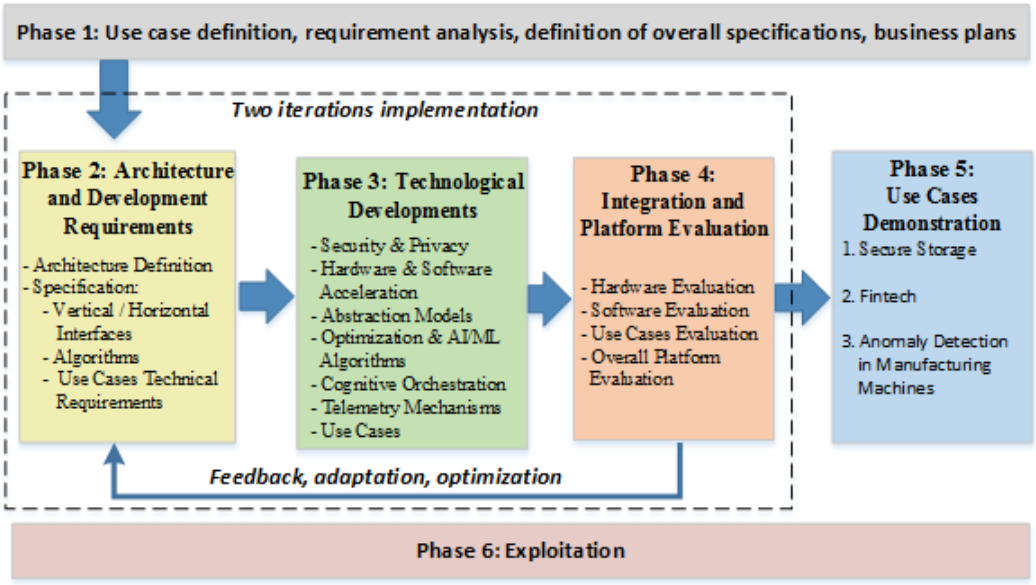


Figure 46: Overall technical development strategy and methodology

Based on the above development strategy, the three main releases (Figure 47) for the integrated SERRANO platform are:

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

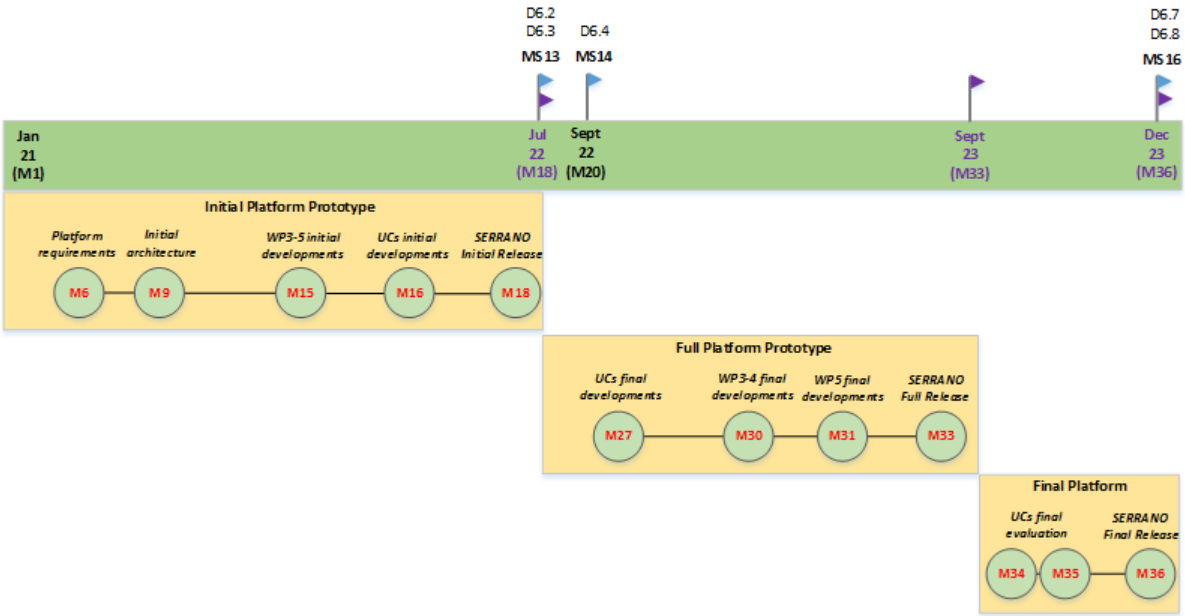


Figure 47: SERRANO development roadmap

The initial platform prototype will be the outcome of the first project development iteration and will provide the partial implementation of SERRANO components. In this version, each component will implement 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) will provide its description and documentation, while its analysis will be used to finalize the SERRANO architecture. Moreover, the initial platform prototype will support the preliminary evaluation of the use cases, reported at deliverable D6.4 “Business, end user and technical evaluation” (M20), that will provide critical feedback for the second development iteration.

The SERRANO full platform prototype will be based on the initial release, and will provide the remaining functionality, which has not been included in the early prototype. As the development in technical work packages (WP3 – WP5) conclude by M31, this release will be provided in M31. The intention is to achieve a fully functional platform resulting from the integration of all project components and provide 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 SERRANO platform (Phase 5) through the demonstration of the three project use cases. The outcomes from the demonstrators will be collected 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 be used for the identification of the critical and high impact components to create the final exploitation plans.

9 Requirements Coverage

In the following table, we present how the initial set of functional requirements, collected, described and categorized in D2.2 “SERRANO use cases, platform requirements and KPIs analysis” (M6) are served by the components of the proposed SERRANO architecture. During the design of the first version of the SERRANO architecture, 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 overall goal was to ensure that the SERRANO platform can realize the desired functionalities.

The updated and final version of the UCs and the platform requirements will be provided in D2.4 “Final version of SERRANO use case, platform requirements and KPIs analysis” (M16) and will be incorporated in the final SERRANO architecture in D2.5 “Final version of SERRANO architecture” (M18).

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-premise 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-premise Gateway, Edge Storage
F_SIR.3	Automatic creation/ selection of storage policies	Essential	Distributed Secure Storage, On-premise Gateway, Resource Optimization Toolkit
F_SIR.4	Secure storage service should maintain data access from browsers	Desired	On-premise Gateway
F_SIR.5	Secure execution of workloads	Essential	Trusted Execution Mechanisms
F_NTFR.1	Discover and monitor heterogeneous resources	Core	Central Telemetry Handler, Enhanced Telemetry Agents, Monitoring Probes, ARDIA models
F_NTFR.2	Dynamically monitor short-lived applications (support serverless architecture)	Core	Hardware Acceleration Abstraction, Enhanced Telemetry Handler
F_NTFR.3	Estimate inter-site and intra-site network characteristics	Core	Enhanced Telemetry Agents, Monitoring Probes, Central Telemetry Handler
F_NTFR.4	Autonomously monitor cloud-native applications over heterogeneous distributed resources	Core	Central Telemetry Handler, Enhanced Telemetry Agents, Orchestration Drivers
F_NTFR.5	Detect critical situations that may lead to application reconfigurations or data migration	Core	Enhanced Telemetry Agents, Service Assurance and Remediation
F_NTFR.6	Exchange events between the components of the hierarchical telemetry infrastructure	Core	Enhanced Telemetry Agents, Data Broker
F_NTFR.7	Zero-touch and data-driven reconfigurability of telemetry components	Core	Central Telemetry Handler, Enhanced Telemetry Agents, Orchestration Drivers
F_NTFR.8	Ability to centralize the monitoring information in a common view or place	Core	Central Telemetry Handler, Enhanced Telemetry Agents
F_NTFR.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	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_SOR.1	Decomposition of applications into independent well-defined tasks	Core	AI-enhanced Service Orchestrator, ARDIA models
F_SOR.2	Description of tasks should support additional information (i.e. metadata, deployment requirements)	Core	ARDIA models
F_SOR.3	Accurate description of heterogenous resources capabilities in a machine processable format	Core	ARDIA models
F_SOR.4	Platform should detect 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

10 Conclusions

Deliverable 2.3 reports on the work performed in WP2, for defining the architecture of the SERRANO platform. The architecture intends to create an intent-driven paradigm of federated infrastructures consisting of edge, cloud and HPC resources.

The deliverable presents the SERRANO-enhanced hardware and software resources, the primary services/components of the SERRANO platform, the multi-layer overall architecture, a set of information flows between components, and the vertical and horizontal components' interfaces. The development roadmap of the SERRANO platform is also presented, along with a discussion regarding the way SERRANO architecture serves the challenging technical and use cases-related requirements.

This document will be further used as a guideline for the technical activities in WP3-5 regarding the SERRANO platform and in WP6 for the development and integration of the UCs. This 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.

The architecture presented in this deliverable will be adapted during the actual development process. The updated and final version of the SERRANO platform architecture will be presented in D2.5 "Final version of SERRANO architecture" (M18).

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] Semiconductor Computer Engineering. EPYC 7742 - AMD. Accessed: 2019-11-04. 2019. url: https://en.wikichip.org/wiki/amd/epyc/7742.

[5] 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

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

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

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

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

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

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

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

[13] 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

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

[15] Docker in swarm mode: https://docs.docker.com/engine/swarm/

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

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

[18] S. Xydis, A. Bartzas, I. Anagnostopoulos, D. Soudris, and K. Z. Pekmestzi, "Custom multi-threaded dynamic memory management for multiprocessor system-on-chip platforms," in ICSAMOS, 2010, pp. 102-109

[19] Y. Sade, M. Sagiv, and R. Shaham, "Optimizing C multithreaded memory management using thread-local storage," in Proceedings of the 14th International Conference on Compiler Construction, ser. CC'05, 2005, pp. 137-155

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

[21] Apache ActiveMQ: https://activemq.apache.org

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

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

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

[25] Diamantopoulos, D., Galanis, I., Siozios, K., Economakos, G., & Soudris, D. (2015). A framework for rapid system-level synthesis targeting to reconfigurable platforms. In Workshop on Reconfigurable Computing (WRC).