Project Title	Artificial Intelligence in Secure PRIvacy-preserving computing coNTinuum
Project Acronym	AI-SPRINT
Project Number	101016577
Type of project	RIA - Research and Innovation action
Topics	ICT-40-2020 - Cloud Computing: towards a smart cloud computing continuum (RIA)
Starting date of Project	01 January 2021
Duration of the project	36 months
Website	www.ai-sprint-project.eu/

D1.3 - Initial Architecture Design

Work Package	WP1 | Requirements & Architecture Definition
Task	T1.3| Architecture Definition
Lead author	Daniele Lezzi (BSC)
Contributors	Hamta Sedghani (Polimi), Matteo Matteucci (Polimi), Andrei Popa (BECK), German Moltò (UPV), Giacomo Verticale (Polimi), Federica Filippini (Polimi), André Martin (TUD), Patrick Thiem (C&H)
Peer reviewers	Danilo Ardagna (Polimi), Enrico Abate-Daga (BECK)
Version	V1.0
Due Date	30/06/2021
Submission Date	30/06/2021

Dissemination Level

X	PU: Public
	CO: Confidential, only for members of the consortium (including the Commission)
	EU-RES. Classified Information: RESTREINT UE (Commission Decision 2005/444/EC)
	EU-CON. Classified Information: CONFIDENTIEL UE (Commission Decision 2005/444/EC)
	EU-SEC. Classified Information: SECRET UE (Commission Decision 2005/444/EC)

AI-Sprint - Artificial Intelligence in Secure PRIvacy-preserving computing coNTinuum, has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 101016577.

Versioning History

Revision	Date	Editors	Comments
0.1	05/05/2021	Daniele Lezzi	ToC definition
0.2	04/06/2021	Daniele Lezzi, Federica Filippini	Sections 1, 2 added, initial version of Scheduling for accelerated devices
0.3	07/06/2021	Germán Moltó	Included section 2.1, A.5. Contributed to 2.3
0.4	10/06/2021	Daniele Lezzi,	Executive summary
0.5	17/06/2021	Federica Filippini, Hamta Sedghani	Review of design space exploration, GPU scheduler and runtime configuration tools sections
0.6	24/06/2021	Federica Filippini, Hamta Sedghani, André Martin, Patrick Thiem	Added section A.2 and review of performance models description Added section 2.4
0.7	25/06/2021	André Martin	Added section A.12
0.8	27/06/2021	Giacomo Verticale	Added section A.13
	27/06/2021	Patrick Thiem	Added section A.14
	27/06/2021	Danilo Ardagna	Review sections 1-2, 3.2, A.1, A.2, A.4
	28/06/2021	Danilo Ardagna	Review sections A.5-7, A.9, A.11-14
	28/06/2021	Matteo Matteucci	Reviewed sections 1.x, 2.x, 3.x, A.X Contributions to 2.2, 2.3 Added sections A.3, A.8, A.10
0.9	28/06/2021	Daniele Lezzi	Section 3 edited
	28/06/2021	Danilo Ardagna	Review of sections 2.2, 2.3, A.3, A.8, A.10
	29/06/2021	Danilo Ardagna	Review of sections 1 and 3.
	29/06/2021	Enrico Abate-Daga	Review of the whole document
1.0	29/06/2021	Daniele Lezzi	Final formatting

Glossary of terms

Item	Description
AI	Artificial Intelligence
CMP	Cloud Management Platform
DevOps	Software development (Dev) and IT operations (Ops)
EDDL	European Distributed Deep Learning library
FaaS	Functions as a Service
GPU	Graphics Processing Unit
IaaS	Infrastructure as a Service
MEC	Mobile Edge Computing
OS	Operating System
TEE	Trusted Execution Environment
TOSCA	Topology and Orchestration Specification for Cloud Applications
UE	User Equipment
TPM	Trusted Platform Module

Keywords

Artificial Intelligence; Edge Computing; Computing Continuum; Programming Models; Runtime management;

Disclaimer

Executive Summary

This document, developed by the AI-SPRINT project, represents an initial version of the platform Architecture deliverable (the final due at M24) that offers an overview of the architectural design.

The focus of this document is the identification of different components and interfaces within the AI-SPRINT architectural building blocks, as well as the different strategies necessary for a successful deployment. The role of each of the blocks is defined, as well as the required interfaces for the communication between blocks.

The architectural choices are built taking into account the different types of requirements imposed by the applications and services running in the framework through the analysis of the three use cases specified in the project proposal and detailed in AI-SPRINT Deliverable *D1.2 - Requirements Analysis*.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Scope of the document

The aim of the AI-SPRINT "Artificial intelligence in Secure PRIvacy-preserving computing coNTinuum" project is to develop a platform composed of design and runtime management tools to seamlessly design, partition and operate Artificial Intelligence (AI)I applications among the current plethora of cloud-based solutions and AI-based sensor devices (i.e., devices with intelligence and data processing capabilities), providing resource efficiency, performance, data privacy, and security guarantees. This document overviews the AI-SPRINT architectural framework introducing the main project assets that will be developed and evolved during the project.

1.2 Target Audience

The Architecture Design Document (initial and final version) is intended for internal use, although it is publicly available. The target audience is the AI-SPRINT technical team including all partners involved in the delivery of work packages 2,3 and 4 but also it serves as reference for the developers of the three use cases of the project.

1.3 Structure of document

This document includes three main parts:

- The **AI-SPRINT Architectural Overview** that provides the description of the main components of the platform including the design tools, the runtime framework, the deployable infrastructure and the security mechanisms amongst all the components.
- The **Integrated Framework** details how the components interact and the different deployment options to support scenarios required by the AI-SPRINT use cases.
- In the Appendix, we provide the details of those possible interaction scenarios and the sequence diagrams with the explanation of the data exchanged.

3. AI-SPRINT Architectural Overview

This section shows the global design of the AI-SPRINT framework, indicating which main functionalities the AI-SPRINT platform will provide for the first iteration of the tools development. For each individual block of the system, we describe its role and relation within the global architecture, as well as, the required interfaces for the communication with other modules.

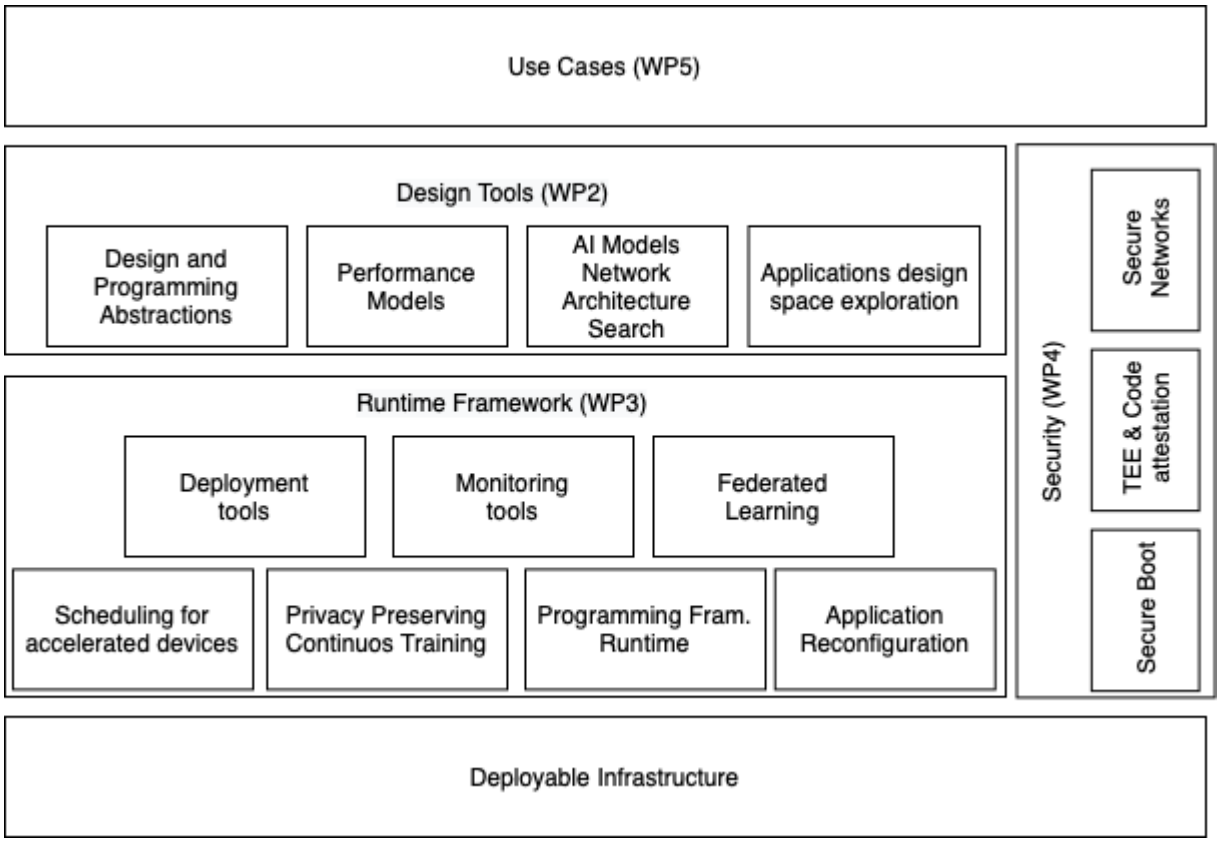


Figure 3.1 - AI-SPRINT Architecture Overview

AI-SPRINT will overcome current technological challenges for the design and efficient execution of AI applications exploiting resources in the edge-to-cloud continuum such as flexibility, scalability, interoperability, security and privacy. Figure 2.1 depicts the high level view of the AI-SPRINT architecture. An AI-SPRINT application is mostly written in Python and it makes intensive use of AI technologies. An application usually comprises multiple components which run across a computing continuum with some components allocated on the cloud, some on edge servers some on AI-enabled sensors (i.e., devices with intelligence and data processing capabilities).

In particular, the AI-SPRINT use cases (see also the *AI-SPRINT Deliverable D1.2 - Requirements analysis*) will be implemented by using the AI-SPRINT design and development tools. The design environment includes **Programming Abstractions** to hide the communications across components and to transparently implement the parallelization of the compute-intensive part of the application, possibly exploiting specialized resources (e.g., GPUs and AI enabled sensors). Applications are also enriched with quality annotations (e.g., data flow rates, application latency, energy constraints) to express performance, accuracy, privacy, and security constraints. **Performance Models** tools automate the AI application performance profiling and identify the

performance model (based mainly on Machine Learning) providing the highest performance prediction accuracy. In particular, **Performance Models** tools will support performance ML model selection and hyper-parameters tuning also considering for the target deployment AI-based sensors. Deep networks might be deployed across computing continua by considering also multiple partitioning options. **AI Models Network Architecture Search** provides solutions to enable developers with limited ML expertise to train high-quality models specific to their needs also in terms of Quality of Service (QoS) requirements. These tools help developers to build a learning as a service solution, that, starting from a training set with labelled training examples (images or temporal data series which are of interest for use cases) will automatically identify the most accurate deep neural network which provides execution time guarantees. Furthemore, for complex applications involving many components, multiple alternative candidate deployments will be evaluated through the **Applications design space exploration** tools maximizing resource efficiency while minimizing the cloud usage cost.

AI-SPRINT applications will be deployed and executed transparently on a heterogeneous architecture, through a set of runtime tools which manage the different components and data of the applications and reacts to system perturbations and requirement changes. The AI-SPRINT runtime environment will include tools to: i) support the continuous deployment of AI applications; ii) support application components concurrent execution reacting to node failures and identifying their optimal placement and resource capacity; iii) trigger automated model retraining, leveraging on solutions for training and retraining at the edge; iv) optimize the scheduling and assignment of accelerator devices among competing training jobs. **Deployment Tools** provide automatic mechanisms to guide the process of configuring computing resources across the computing continuum and provide a cloud-edge orchestration enabling the automatic deployment of AI application models and components, without manual provisioning. The **Programming Framework Runtime** is in charge of supporting concurrent code execution, automatically detecting and enforcing the data dependencies among components and spawning parallel tasks to the available resources, which can be nodes in an edge cluster or clouds. **Privacy preserving continuous training** allows AI applications to update models in order to capture changes just in weight matrices and internal state initialization of the model, or in the model structure because of retriggering of the AI model architecture search. Edge training and retraining in AI-SPRINT will leverage **Federated Learning** algorithms where different learners exchange only part of the information which can be extracted from the data in order to improve a model globally, but without making explicit the data used to train it. The efficient use of GPUs for the training of neural networks will be addressed by proposing advanced techniques of **Scheduling for accelerator devices** to solve the joint resource planning (i.e., how many GPUs to assign to a training job) both for private and public clouds. Application components will be continuously monitored by a **Monitoring Platform** which will be able to gather metrics at every layer of the cloud-edge stack.

AI-SPRINT will provide tools for the deployment of compute instances enabled with **Trusted Execution Environments (TEEs)**, that allows application entities to be orchestrated in a way that only trusted parties are allowed to establish communication channels with each other through end-to-end encryption, and code attestation and verification mechanisms to ensure that only the correct code/binary is being executed. The framework will also integrate **Secure Booting** mechanisms to ensure that the OS where applications are running can be attested to provide an extra level of security. The AI-SPRINT **Secure Networks** component will leverage network programmability to manage the setup of virtual networks, to steer secure tunnels through them, and will ensure that the network paths comply with the security policies defined at design time including when traffic is rerouted in case of failure.

Table 1. 2 summarizes the WP-level responsibilities of the different components of the architecture, the lead maintainer and the major contributors.

Tool	WP	Task	Lead Maintainer	Major contributors
Design and Programming Abstractions	2	T2.1	BSC	BSC, POLIMI, TUD,UPV
Performance Models	2	T2.3	POLIMI	POLIMI, UPV, BSC
AI Models Network Architecture Search	2	T2.2	POLIMI	POLIMI, BSC, TUD
Application Design Space Exploration	2	T2.4	POLIMI	BSC, UPV
Deployment Tools	3	T3.1	UPV	UPV, TUD, POLIMI, BSC, BECK, C&H, 7BULLS
Monitoring Tools	3	T3.3	7BULLS	7BULLS, TUD, BSC, UPV
Federated Learning	3	T3.4	POLIMI	POLIMI, BSC, TUD, BSC, UPV, 7BULLS
Scheduling for Accelerated Devices	3	T3.5	UPV	UPV, POLIMI, 7BULLS, TUD
Privacy Preserving Continuous Training	3	T3.4	POLIMI	POLIMI, TUD, BSC, UPV, 7BULLS
Programming Framework Runtime	3	T3.2	BSC	BSC, POLIMI, UPV
Application Reconfiguration	3	T3.2	C&H	C&H, POLIMI, TUD
TEE & Code Attestation	4	T4.2	TUD	TUD, UPV
Secure Boot	4	T4.4	TUD	TUD, C&H, POLIMI, UPV
Secure Networks	4	T4.3	POLIMI	POLIMI, TUD

Table 3.1 - AI-SPRINT WP Level Architecture

3.1 Deployable Infrastructure

The AI-SPRINT architecture is built upon a flexible computing substrate provided by cloud and edge-based technologies.

Concerning the former, cloud computing allows the on-demand provision of virtualized resources to support both the computing and storage requirements from the use cases. Different cloud service models will be adopted. First, Infrastructure as Service (IaaS) provides the ability to deploy custom Virtual Machines (VMs) that are specifically configured with the Operating System (OS), libraries, runtimes and applications that use cases require. This is achieved through the use of Cloud Management Platforms (CMPs) such as OpenStack and OpenNebula, which provide resource management capabilities featuring both API-based and web-based access mechanisms for programmatic and end-user access respectively.

In order to facilitate deterministic repeatability of virtual infrastructure deployments, we are adopting an Infrastructure as Code (IaC) approach that is being widely adopted by the industry. IaC allows to define application architecture using high-level recipes and resort to automated procedures both for virtual computing resources provision and automated configuration of said resources. A DevOps approach will be adopted in order to achieve deterministic virtual infrastructure deployments across multiple cloud back-ends. This is exemplified by open-source tools such as Ansible, Puppet and Chef, which allow the definition of templates that describe the desired state of the virtual infrastructure and provide means to enact its automated configuration for the sake of compliance. Second, Functions as a Service (FaaS) allows to create functions coded in a certain programming language and which are triggered in response to certain events (such as an HTTP request to an API endpoint or a file upload to an object-storage system). These function invocations are executed on computing resources that are automatically managed by the provider so that elasticity no longer becomes the main concern of the application developer. These functions also use a fine-grained cost model, typically in the order of milliseconds of execution, as exemplified by public cloud services such as AWS Lambda. The FaaS computing model has experienced tremendous growth in the last years leading to the surge of different open source FaaS platforms that aim to replicate the functionality provided by their public cloud FaaS services counterparts, but on a Container Management Platform (CMP) such as Kubernetes.

Regarding edge computing, the computing infrastructure requires to be built on low-powered devices that prioritize energy efficiency over high-end computing capabilities. Examples of these devices include Raspberry Pis, which are single-board computers that were initially targeted to promote computer science in developing countries but that have found important niches in the field of edge computing. We plan to adopt these boards, possibly adopting accelerators, to create portable computing devices with varying computing capabilities (from Raspberry Pis to even small clusters) that can satisfy the computing mobility requirements of some use cases (e.g., agricultural). In AI-SPRINT we assume that these edge devices support the execution of containers in order to bootstrap the virtualized infrastructure and configuration through automated means. To this aim, we are adopting efficient strategies of resource allocation on these devices, such as those provided by minified Kubernetes distributions, exemplified by K3S[1] and MicroK8s[2].

AI-SPRINT builds on these foundational blocks to support a flexible and customizable deployable infrastructure that spans across the computing continuum including edge devices, on-premises Clouds and public Clouds. This flexibility, combined with the adoption of open standards, well-known tools from the community and best practices in infrastructure deployment, will facilitate the support of the Use Cases.

3.2 Design Tools

The objective of the AI-SPRINT Design Tools is to provide a layer that abstracts the applications from the underlying computing resources, being these edge resources or cloud servers in such a way that the application developer only needs to focus on the actual algorithm and application logic. On the other hand, interfaces for easing the integration of developed applications with the runtime system will be developed.

In particular, the AI-SPRINT developer is provided with a framework to design AI applications using abstractions to specify quality parameters and to express the resource requirements of the components of AI distributed applications. Performance models are used to predict execution times for the different parts of AI applications in a given deployment and to drive the mapping of components on processing elements.

[1] K3S. https://k3s.io/

[2] MicroK8s. https://microk8s.io/

Security policies are available in the definition of AI applications to run on different deployments as edge resources or cloud premises.

Tool	Motivation	Innovation
Design and Programming Abstractions	AI applications considered in AI-SPRINT are composed of different components that combine different approaches adopting popular frameworks such as PyTorch, Tensorflow, Scikit-learn. These components have to be executed on the edge-cloud continuum considered by the project, with a combination of CPUs and GPUs based distributed nodes. On top of this, users have to deal with big datasets on distributed resources. Furthermore, applications must be configured using policies that define which services are allowed to communicate with each other as well as what files and communications channels should be encrypted using TLS. Besides configuring the encryption and access policies, the secure policies definition also provides means of coordinating secret sharing and provision as well as attestation of applications to ensure that an application is indeed genuine and running on trusted hardware.	• Offer high level interfaces to several AI/ML algorithms based on PyCOMPSs and standard frameworks. • Concentrate on the application development and rely on the runtime. • Automatic parallelization of the code. • Partition the data, and possibly the deep network models, to optimize the execution. • Introduce performance parameters for the allocation of tasks to computing continuum resources and security & privacy annotations for data allocation and processing. • Define applications following a FaaS model.
Performance Models	Once an AI application is designed, performance models can help to anticipate the performance of the application components before the production deployment or throughout revision cycles. Example of questions that can be answered by performance models include: how many resources (GPUs, memory, CPU threads, etc.) will be required to achieve a given performance target (e.g., training job will end within a due date? the inference time of an AI	• Support profiling, given multiple allocations at the computing continuum. • Provide novel performance models based mainly on machine learning, experiment design methods and, possibly, queueing networks. • Automate components performance models training.

	component will be lower than a given threshold ?). Solutions to predict AI application components performance under different configurations and deployment settings at the full computing continuum stack will be developed.	
AI Models Architecture Search	The scarcity of AI experts, the need for the design of customized network architectures, the need for a long and tedious hyperparameter tuning, make the development of novel AI models a time and resource consuming effort. To reduce the time to market of novel models customized for the specific use case and for the specific computing continuum under evaluation, we wil design a tool to identify and automate the search of the most accurate deep network which complies with QoS constraints starting from labelled data. The possibility of specifying a QoS is what makes AI-SPRINT neural architecture search different from the competitors and it is a unique feature of the AI-SPRINT framework.	• Include runtime performance and QoS as optimization criteria in the search of the best neural architecture. • Consider the multiple layers of the computing continuum when optimizing and searching for an optimal architecture which it will be possible to deploy on the design target. • Integrate architecture search with cloud-edge model partitioning by allowing the search algorithm to be aware of the network partitioning when optimizing its architecture. • Multi-objective space exploration and Pareto optimal solutions. • Performance estimation as a driver of search process. • Model pruning.
Application Design Space Exploration (SPACE4AI-D)	Once an AI application approaches the final deployment stage, it becomes increasingly important to tune its performance. This is a time-consuming task, since the number of different configurations can grow very large. Tools are needed to guide this phase. The ultimate goal of the *Application Design Space Exploration tool* is to explore automatically multiple candidate deployments for the AI application components to identify an optimal initial deployment which satisfies performance constraints.	• Define ad-hoc algorithms (based on random greedy, local search and other meta-heuristic methods) tailoring the peculiarities of AI applications (e.g., multiple versions corresponding to different partitions of the same deep network deployable on edge resources or cloud) to quickly find good designs given QoS constraints.

Table 3.2 - Design Tools: motivations and innovations

The design framework will also provide means to define applications components as microservices and event-driven functions that can be dynamically orchestrated at runtime, according to the FaaS model. The aim of the design abstractions is to provide high-level annotations to specify QoS constraints and code dependencies and to introduce performance parameters for the allocation of tasks to computing continuum resources and security & privacy annotations for data allocation and processing.

The design time framework will support the composition of ML, DL and AI applications primarily adopting the PyCOMPSs programming model and tools (e.g., dislib library), enabling the development of complex and dynamic workflows, composed of pure computational parts, classical data analytic runs and ML/DL methods. At the same time, the adoption of popular AI/ML frameworks will be guaranteed and in all the cases applications will be enriched with annotations to specify non-functional properties constraints, security and privacy policies, and application candidate target deployments. A number of ML algorithms implemented with PyCOMPSs are already available as a Distributed Computing Library (dislib[3]) inspired by Scikit-learn, which eases the task of developing applications providing a common interface in all algorithms.

Furthermore, as a parallel framework, COMPSs exploits inter-node and intra-node parallelism and executes TensorFlow tasks as external processes. The design layer will be based on the PyCOMPSs programming framework, but integration with popular frameworks as PyTorch, TensorFlow, Keras will be also provided. The main benefit of the integration of the AI-SPRINT programming model is the reduction of the execution time of the training and inference processes by exploiting the inherent data parallelism of input data. To do so, the input dataset is distributed over the different targeted nodes and each node is in charge of training the model with the assigned dataset part. Later, the different models trained in isolation are combined thus requiring to exchange either the model (parameters) or the parameter's gradients between the nodes.

Furthermore, AI-SPRINT will develop novel solutions to model the performance of AI applications running at the full computing continuum stack. **Performance models** can serve as a fundamental building block for cloud service providers hosting AI services in the cloud. Indeed, performance modeling enables cloud providers to estimate applications performance based on provisioned resources and to possibly manage effectively data center infrastructures at runtime. Moreover, based on the performance modeling feature, cloud providers can offer what-if analysis for customers to perform trade-off analyses between cost and application execution time. For example, customers might be willing to pay an additional 10% to speed up their application by two if there is such a choice but choose not to do so if they have to pay twice more for 10% performance improvement. Without performance modeling and prediction tools, it is hard to obtain such information. The AI-SPRINT performance modelling approach will be mainly based on ML. ML-based performance models will be developed to predict the execution time of AI applications running on heterogeneous multi-devices systems composed of IoT & AI enabled sensors, edge and cloud resources. The tools will provide support and automate AI applications performance profiling. Solutions to identify the ML model providing the highest performance prediction accuracy supporting model selection and hyperparameters tuning will be also developed. AI-SPRINT solutions will be based on the aMLLib library (https://github.com/eubr-atmosphere/a-MLLibrary) which will be enhanced to improve the hyperparameters tuning phase.

To allow a fast development of targeted AI models also to non AI Expert people, but also to support AI Experts in their work, a module for model search via Neural Architecture search (NAS), will be implemented. The AI-SPRINT NAS module will take into account desired QoS both from a machine learning perspective (e.g., accuracy, precision, recall, etc.) and a performance perspective (e.g., latency, memory, power consumption, etc.). To allow for this, performance models will be used in order to predict the future performance of a given architecture even before deploying it and running it on the real platform. By the use of performance models,

[3]https://dislib.bsc.es/en/stable/

it becomes possible to speed-up the search of Pareto optimal solutions disregarding those architectures which are not considered as suitable with respect to the AI model requirements.

Application design space exploration will leverage the SPACE4AI-D tool (*System PerformAnce and Cost Evaluation on Cloud for AI applications Design*). SPACE4AI-D tackles the component placement problem and resource selection in the computing continuum at design time, dealing with different AI application requirements in order to effectively orchestrate heterogeneous edge and cloud resources. The tool will leverage some efficient meta-heuristic algorithms such as random greedy, local search, tabu search and so on, in order to explore automatically multiple candidate deployments for the AI application components to identify the placement of minimum cost across heterogeneous resources including edge devices, cloud GPU-based Virtual Machines and FaaS solutions, under QoS response time constraints.

3.3 Runtime Framework

The objective of the AI-SPRINT Runtime Framework is to support the automated deployment and monitoring of customized virtualized resources across the computing continuum infrastructure. This includes the orchestration of computations on that provisioned infrastructure in order to support the training of AI models and also exposing the trained models as a service using a FaaS paradigm for inference. Moreover, load variations may induce resource saturation or underutilization, which have a possibly strong impact on components response times and execution costs. The Runtime Framework has therefore the goal of adapting the component placement to account for such events, by migrating components from edge to cloud and vice versa, by scaling the amount of cloud resources and/or by changing DNN models partitions configuration.

Our flagship product in this area is the **Infrastructure Manager** (IM)[4], an open-source tool to deploy customized configurable application architectures described using the TOSCA (Topology and Orchestration Specification for Cloud Applications)[5] standard on multiple Cloud back-ends. These includes widely used on-premises CMPs such as OpenNebula and OpenStack; public Cloud providers such as Amazon Web Services, Microsoft Azure and Google Cloud Platform; European Cloud infrastructures such as the EGI Federated Cloud and commercial providers such as Open Telekom Cloud and Orange.

Two previous developments have been onboarded in AI-SPRINT to support the FaaS requirements from use cases. On the one hand, **SCAR**[6] (Serverless Container-aware ARchitectures) supports compute-intensive functions packaged as Docker images on top of AWS Lambda, allowing automated delegation of function execution into AWS Batch, a service that deploys elastic clusters on top of Amazon EC2, the IaaS managed service provided by Amazon Web Services (AWS). This approach combines the benefits of high elasticity provided by AWS Lambda with the unbounded computing capacity provided by AWS Batch, allowing to create data-driven serverless workflows typically aimed at file processing.

On the other hand, **OSCAR**[7] (Open-source Serverless Computing for Data-processing Applications) supports the very same computing model offered by SCAR but within an on-premises CMP. OSCAR consists of an elastic Kubernetes cluster, which is dynamically deployed on all the Cloud providers supported by the IM. The cluster can grow and shrink in terms of the number of nodes, thanks to the **CLUES**[8] elasticity system, which inspects the status of the Kubernetes cluster to instruct the IM to provision or terminate the nodes in order to self-adapt to the current and expected workload. The use of a minified Kubernetes distribution running on Arm-

[4] Infrastructure Manager (IM). https://www.grycap.upv.es/im

[5] OASIS Topology and Orchestration Specification for Cloud. https://www.oasis-open.org/committees/tosca/

[6] SCAR. https://github.com/grycap/scar

[7] OSCAR. https://github.com/grycap/oscar

[8] CLUES. https://www.grycap.upv.es/clues

based processors, such as those provided by Raspberry Pis, allow to execute OSCAR clusters for the in-the-field inference of previously-trained Deep Learning models via a FaaS. These computing resources can be supplemented with on-premises CMPs and even public clouds in order to adapt to increased computing needs.

Tool	Motivation	Innovation
Deployment tools	Need to provide efficient tools to perform automated provision and configuration of virtual resources, supporting complex application architectures. The adoption of open standards for application architecture description is mandatory to foster sustainability. Also, high-level recipes for automated infrastructure configuration and application installations need to be adopted to guarantee determinism in distributed and multi-tenant infrastructures. These recipes need to be made publicly available to foster widespread adoption beyond the lifetime of AI-SPRINT	• Infrastructure as a Code combined with standards like TOSCA uses high-level recipes to define application architectures whose resources are automatically provisioned and configured. This introduces repeatability and deterministic deployments. • Open code repositories and artifacts such as GitHub, Docker Hub and Ansible Galaxy are populated with ready-to-be-used components for automated application deployment across the cloud continuum (edge, on-premises clouds and public clouds). • Multi-modal interfaces are provided for automated application deployment including command-line interfaces, REST APIs and web-based graphical user interfaces to accommodate several categories of users. • A Continuous Delivery (CD) approach is adopted to cope with application changes that need to be distributed across disparate computing infrastructures, building on automated procedures for creating deployment artifacts, such as GitHub Actions and Automated Builds in Docker Hub.
Monitoring tools	Without knowing the overall state of all systems components, one cannot adjust their parameters and execute necessary actions when failures occur. That is why there is a need to provide subsystem which will be responsible for: (i) collecting monitoring data (metrics, statuses, health checks) from all subsystems, (ii) provide the possibility to analyse this	• Improving monitoring data fetching from temporarily off-line subsystems. • Providing seamless integration with client applications. • Extending algorithms for anomalies detection.

	datastream, and (iii) notify controlling subsystems about spotted anomalies. The whole monitoring solution should also be as much "invisible" as possible for monitored applications, allowing developers to concentrate on algorithms and not on integration issues. Monitoring must be also well-integrated with deployment tools, providing advanced data gathering functionality without the need of additional adjustments by system administrators or developers. In case of subsystems which temporarily work off-line (without permanent connection to monitoring infrastructure), there is a need to provide mechanisms which will buffer data for a long period of time (hours, days).	
Programming framework runtime	The execution of AI applications on Edge-Cloud infrastructures requires a runtime that properly assigns the different components on the available nodes, also distributing the data. The runtime leverages the embedded computing resources of each node to host the execution of functions in a service manner and generates hybrid workflows, composed of atomic and continuous processing tasks, achieving distribution, parallelism and heterogeneity across edge/cloud resources transparently to the application developer. The adoption of such a distributed model for executing the applications requires to isolate the applications from infrastructure specificities and (of course) to avoid being locked-in a specific platform.	• Concentrate on the application development and rely on the infrastructure management by the serverless platform. • The programming framework runtime parallelizes the execution of the different parts of the applications that can be invoked in a FaaS way according to the QoS constraints. • The runtime is able to schedule the tasks on both edge and cloud devices, orchestrating the execution and leveraging on fault tolerance mechanisms to react to the dynamicity of the edge. • Adaptation includes also the possibility to migrate tasks from cloud backed to mobile devices (Android) and vice versa.

Federated Learning	When multiple parties want to jointly train a machine learning model without the exchange of sensible or private data the use of federated learning will make this possible. In the federated learning approach, data are expected to reside as close as possible to the sources which have generated them and they reach, at most, the edge part of the continuum without leaving the owner borders. AI-SPRINT will allow the seamless and secure implementation of different approaches for federated learning, either exchanging full models or exchanging model gradients. To protect from privacy attacks techniques for differential privacy will be implemented not to disclose information about individuals or specific records.	• Federated learning used to minimizes information exchange and preserve data privacy integrated in the runtime environment and into the programming abstraction. • Losses and gradients computed as close as possible to the data sources taking into account model partitioning between edge and cloud. • Differential privacy enhancement of exchanged gradients to further protect the privacy of the individual or specific records which the owner does not want to disclose.
Scheduling for accelerated devices	DL models are usually trained on hardware accelerators (e.g., GPUs) achieving on average 5-40x speed-up wrt. CPUs. The ability to optimize the infrastructure utilization and process the workload with high efficiency under power constraints is critical for cloud data centers subject to power consumption quotas. On the other hand, public clouds GPU-based VM time unit cost is 5-8x higher than high-end CPU-only VMs. While there are advanced systems to manage virtual Web application workloads, there are few solutions for GPU-based systems. The goal of the *Scheduling for accelerated devices component* is to transparently support GPGPUs systems sharing in private/public cloud and edge-based environments to maximise system usage/minimise	• Define TOSCA extensions for hardware accelerators. • Use GPU virtualization (rCUDA). • Support dynamic reconfiguration and efficient access to shared accelerators • Enable remote access to GPUs from edge resources. • Develop novel fast-heuristics for accelerators scheduling at cloud backends (private and public) and edge servers, matching accelerators capabilities with training jobs requirements.

	operational costs while easing the programming models. Solutions to support elasticity at the level of GPGPUs also through disaggregated hardware technologies will be developed and integrated with advanced schedulers able to identify the optimal training jobs execution order, GPU number and partitioning among running jobs providing also upper bounds on the training time.	
Privacy preserving continuous training	When deployed on the field AI models might experience some drift in the observed data or might need an update because novel data have been accumulated in the meanwhile. This continuous training requires models to be updated periodically or because of a triggering event. This continuous update of models has a clear impact on the existing applications which need to be updated securely with minimal impact on the system overall. Because of this, two different means of updates are foreseen, one which leverages on the straightforward weight update and one which re-deploys from scratch the whole architecture, possibly re-partitioning it among different devices.	• Seamless, secure, privacy preserving model weights update. • Application redeployment after re-partitioning in case the new model requires a different optimization or a different set of resources. • Automated triggering of the update upon designer defined conditions.
Application reconfiguration (Krake and SPACE4AI-R)	The optimal solution identified by the *Application Design Space Exploration* tool needs to be periodically reevaluated in order to account for load variations. These can, indeed, lead to resources saturation or underutilization, having a possibly strong impact on the components response times and the costs predicted at design time. This goal is fulfilled by the *Application reconfiguration tool*, which, first of all, evaluates the current status	• Develop ad-hoc algorithms to quickly react to load variations at runtime, adapting the components configuration and the resource allocation to the new conditions, in order to optimize execution costs given QoS constraints. • Support runtime migration of stateless and stateful application components across heterogeneous devices (edge and cloud).

	of the system in terms of incoming loads and components response times. Then, on the basis of the collected information, it determines the new optimal solution, specifying the components configuration and the relative deployment on the available resources, optimizing components response times and execution costs.	

Table 3.3 - Runtime Framework: motivations and innovations

The runtime framework will provide several interfaces for provisioning virtual infrastructure to satisfy different user profiles. A REST API for programmatic access will facilitate application integration. A web-based GUI (Graphical User Interface) will allow end users with limited technical skills to self-provision virtualized infrastructure for training models on specific infrastructures customized with certain software and hardware requirements. A CLI (command-line interface) will allow the savvy user to efficiently interact with the provided services in order to manage the lifecycle of dynamically provisioned virtual infrastructures.

The benefit of the runtime framework is to provide the "Design Tools" with the ability to deploy the required computing infrastructure and perform its automated configuration with the precise software artifacts in order to satisfy the user requirements. A wide range of customized virtual infrastructures are envisioned that include, but are not limited to: Kubernetes clusters configured to support accelerated training using GPUs; Monitoring infrastructure, automatically deployed using a DevOps approach for the sake of repeatability; Virtual Machines with GPUs offered as a service using **rCUDA**[9] for remote acceleration of AI model training; OSCAR clusters to support FaaS for compute-intensive model inference.

AI-SPRINT also provides a programming framework runtime based on COMPSs that thanks to the integration with application reconfiguration solutions and to the continuous deployment service support the scheduling of dynamic workflows; these workflows can change their behaviour during the execution (i.e., according to the partial results of the application), to adapt application and react to changes in the execution environment at large. Resource allocation and workflow tasks deployment can be changed in real-time; tasks can be migrated from cloud to edge servers and devices simplifying the provisioning and management of AI applications lifecycle, including support (remote access) to edge and accelerator devices. Developers can write AI/ML applications that will be orchestrated adopting a FaaS paradigm (which supports the execution of transient stateless functions) for the most effective and seamless use of the continuum resources.

For what concerns component application migration, **Krake** (https://gitlab.com/rak-n-rok/krake) will be used. Krake provides a central point for managing component applications at each level of the computing continuum system. Furthermore, it ascertains the "best" level of resource usage based on user-defined parameters and re-evaluates the deployment periodically. This leads to an automation in increasing or decreasing resources. While Krake currently supports only infrastructure metrics (e.g., resource utilization, hardware energy efficiency), it will be extended to support performance-based application metrics and enable optimal decisions on how many resources (e.g., total number of Docker/kernel containers) to allocate to provide latency guarantees (e.g., latency below 100 ms). The goal is to optimally manage edge resources, which often are characterized as devices with low resource capacity to meet application QoS constraints. The

[9] http://www.rcuda.net

allocation of components to resources is based on various criteria (for example battery level, network latency, availability of accelerators to speed up component execution, etc.). While Krake will provide the mechanisms to support components migration to the Application Reconfiguration module, intelligent decisions that will trigger component migration/cloud resource scaling/change in a DN partition will be made by the **SPACE4AI-R** (*System PerformAnce and Cost Evaluation on Cloud for AI applications Runtime*).

Finally, long running training jobs will be supported by the **GPU scheduler**. This receives job submissions, in the form of Docker containers, and determines the best scheduling and component allocation to minimize costs while meeting deadline constraints. The list of submitted jobs, together with their characteristics in terms of expected execution times (collected through profiling) and deadlines, and a description of the system with all the available resources, are provided as input to the GPU scheduler. Based on these information, the GPU scheduler determines which jobs should be run and the type and number of GPUs that should be assigned to them, in order to minimize energy costs (in the case of private GPU-accelerated clusters) or execution costs (in the case of public cloud), while meeting the deadline constraints. Disaggregated hardware architectures and remote GPUs can be accessed relying on rCUDA.

Data describing state and working parameters of all subsystems will be gathered by the **Monitoring** subsystem. It will preserve all collected time series metrics and allow system administrators and other subsystems to perform various actions. In particular the Monitoring infrastructure will provide the possibility to create a self-healing system, which will be able to adapt to changing environmental conditions or detected anomalies and failures. Thanks to the advanced tools that will be developed, the Monitoring subsystem will also help analysing behaviour of the whole system in time and to find causes of bugs or failures.

Data privacy is a first class citizen in the AI-SPRINT framework and thus tools for distributed computing and infrastructure management will be used to implement **Federated Learning** algorithms which are in charge of training models in a distributed fashion having data, errors, and gradients computed as close as possible to their source. The Monitoring infrastructure will be used to **trigger the retrain of models** and their update in the infrastructure which could go from simple weights update, if no major architectural change has happened, to full application redeployment in case a novel more effective partitioning of the model has been found.

3.4 Security and Privacy

The tool we use as a foundation for secure and privacy data processing in the context of AI-SPRINT is SCONE. The framework provides the possibility to run applications in so called enclaves, i.e., trusted compartments such that no adversary or even an inside attacker with root privileges can access or compromise the data as well as the application code. This is achieved by using so-called Trusted Execution environments such as Intel SGX. Although SCONE currently provides only support for Intel SGX, the objective within this project is to extend the existing framework to support other TEE vendors such as AMD and Arm which are also typically found in the domain of edge devices.

The benefit of using SCONE as a foundation for secure computation is the following: It provides transparent encryption for network connections as well as file systems through so called encrypted volumes. Furthermore, the framework aims at minimizing the developers effort, hence, in most cases no recompilation for the application will be needed. This will greatly speed up the adoption process of this approach as only prebuilt Docker container images will be used.

The framework is complemented by the so-called Configuration and Attestation Service (CAS). This entity is responsible for verifying if the application code has been tampered with while or before being loaded into the trusted compartments through a so called attestation process. Once the application has been attested to run on genuine hardware and no modifications occurred, CAS will provision the application through secure channels with the necessary secrets and certificates needed to access files and to establish connections in a

secure manner. The approach follows the principles that no humans should get into the possession of secrets, hence, certificates and key pairs only needed to establish connections between restful services will be solely generated and managed within CAS. The CAS instance itself is also running in an enclave such that the generated secrets will never be exposed to a human or third party entity.

The AI-SPRINT framework is designed to work well with Mobile Edge Computing (MEC) platforms based on 5G access. AI-SPRINT considers scenarios in which the User Equipment (UE) connects to a private 5G network to access a private MEC platform or in which the UE connects to a public 5G network and accesses a MEC platform via Local Breakout. The AI-SPRINT Secure Network (ASSN) component is responsible for implementing granular firewall policy control to restrict the UEs to access only the authorized MEC services and vice-versa. The ASSN component consists in a logically centralized Controller and in one or more Data Plane nodes. The Controller has the role of identifying the authorized traffic streams, correlating the UE identity, the traffic endpoint at the exit of the 5G network, the service entry point in the MEC, and checking its authorization in the policy database. Then, the Controller configures the relevant Data Plane nodes to allow the authorized traffic and prevent any other unauthorized traffic.

The AI-SPRINT edge computing platform receives data from IoT devices and can provide access to the collected data to multiple users, who can be entities that use the data to perform predictions, they can be entities that use the data to train models, or they can be data aggregators. The AI-SPRINT access control component makes it possible to automatically authorize access without the need of a central component. Instead, the authentication of the entity requesting the data and the authorization to access a specific piece of data are managed using a smart contract executed on the Ethereum blockchain.

In terms of secure booting mechanisms, in AI-SPRINT it will be ensured that every possible hardware as well as VM instances are secure Boot-enabled. In addition, hardware root-of-trust security such as TPM will be leveraged. Keylime or a similar system could provide a good entry point into TPM-based secure boot mechanisms. At the very least, these systems will be implemented in the provided GPU nodes. Furthermore edge cloud instances as well as VMs should also automatically be securely booted. Therefore, a separate test procedure is included. The state of the art suggests that IoT devices should also be securely booted in the future by any means necessary to implement security measures across all levels of data processing.

Tool	Motivation	Innovation
Trusted Execution Environments	Traditional isolation techniques such as provided through operating systems by having multiple users etc. and virtual machines as typically used in cloud environments is not sufficient as root users like system administrators of a cloud node can still inspect memory and gain access to credentials by creating memory dumps. Trusted Execution Environments allow to maintain confidentiality and integrity of code and data as an application can be run in an encrypted and isolated memory region called enclave such that no other users including users with	Utilizing Trusted Execution Environments requires adding certain instructions to an existing application in order to load the application code in such a trusted compartment for an isolated execution. While it is possible to integrate the publicly available Intel SGX SDK, it requires a lot of developer effort as interaction between the system calls must be manually instrumented. Furthermore, each CPU vendor provides its own kind of TEE which requires different instruction and manual adoption for programs. The goal of AI-SPRINT is therefore to develop an orchestration framework which enables running unmodified code in TEEs of various vendors such as Intel SGX, ARM Trustzone etc.

	root privileges can read or manipulate the data. However, using Trusted Execution Environments such as Intel SGX requires a certain developer effort to utilize the new instructions provided by processor vendors in order to harness these new properties.	The challenge to achieve this goal is to provide the same level of protection to the end-user regardless of the different properties and guarantees each CPU vendor with his technology provides. This requires a deep analysis of the provided functionality of each vendor and implementation effort to harmonize the different approaches.
Secure Boot	Ensuring the integrity of the framework against malicious attacks and unauthorized updates and guaranteeing to boot the correct operating system (OS).	The integration and use of Secure Boot paired with hardware root-of-trust measures such as TPM is mandatory for AI -SPRINT and will be enabled on servers. This adds an additional layer of security to the platform. In addition, the SCONE code base will provide TEE support for GPUs. Secure Boot itself is a security enhancement, but also suffers from several security vulnerabilities that will need to be addressed in the future. It will not be possible to address all security vulnerabilities within AI-SPRINT, but the challenges that need to be addressed are the identified policies for Secure Boot (e.g., automated upgrade procedures) and TPM usage, as well as numerous configuration options that should be implemented during automated migration or hosting of edge and cloud nodes and VMs. In addition, current efforts in Secure Boot research suggest that Secure Boot procedures should also be implemented in IoT instances. Another challenge would be to implement a procedure that checks if Secure Boot itself is enabled, even if the TPM confirms it.
Secure Networks	In a 5G Mobile Edge Computing scenario, an attacker having access to a user device can try to move laterally, trying to gain access to an unauthorized service, or to an authorized service claiming to be a different user.	The component should perform policy checking at wire speed. AI-SPRINT will achieve this goal by using the P4 language for programmable network data plane. This will make it possible to deploy the component both on lower-end virtual switches, for smaller edge scenarios, and on high-end programmable hardware for high performance scenarios.
Access Control to IoT Data	Applications that need to access data collected from IoT devices often require very low latency and	Addressing the needs of businesses to deliver their IoT data to customers at the territorial level, while guaranteeing the

	high availability. Additionally, many businesses want to be able to sell the data they collect from IoT devices (e.g., sensors readings) to customers or other businesses that operate in the same territory, where the data is actually relevant.	transparency and auditability of enforced authentication and access control policies thanks to the use of blockchain technologies. This is achieved through the implementation of a smart contract deployed on the Ethereum blockchain, which is then used to provide authentication and access control capabilities to a web API running in the edge node.

Table 3.4 - Security Policies: motivations and innovations

4. Integrated framework

4.1 Detailed architecture

In this section we provide a more detailed description of the architecture, depicted in Figure 4.1, explaining the interactions amongst components and the possible deployments to support the realization of the use cases.

Applications in AI-SPRINT are going to be composed of different parts following the COMPSs task programming mode as baseline; such programming model is based on sequential development in which the developer is mainly responsible for: i) identifying the functions to be executed as asynchronous parallel tasks; ii) annotating them with standard Python decorators. Existing code based on other frameworks (Tensorflow, PyTorch, Keras etc) can be also executed as external applications, splitting for example the data and thus exploiting the parallelism. To this aim, the EDDL library[10], a library developed within the context of the DeepHealth European Project, can be combined with dislib and PyCOMPSs for supporting distributed and federated training. Annotations will be extended to allow predicating on components performance and to specify constraints on the target deployment. For example, an image processing task can be automatically allocated by the design tools on a specific edge server according to the application performance constraints (e.g., processing time less than 30 ms). In order to enforce a high-security level for the execution of tasks, another annotation will force the code to run in a secure enclave using SCONE to secure the architecture runtime (security by design approach).

Applications will be described as OASIS TOSCA templates describing the topology of their components and their software dependencies. Multiple alternative candidate deployments will be evaluated using SPACE4AI-D, an automated tool to derive, given the QoS constraints, optimal component placement maximizing resource efficiency while minimizing the cloud usage cost.

At the level of the infrastructure, IM is used to dynamically provision the required components for the execution of the applications and the AI-SPRINT framework. AI-SPRINT will use Docker containers to automate deployment and provide the level of isolation needed to enforce performance constraints with minimal overhead. Different target devices will be used as resources, ranging from edge Raspberry PI with Minified Kubernetes to OpenNebula clusters with accelerated nodes to perform inference jobs, while batch jobs for training of the models will be executed on both private and public clouds. The COMPSs Runtime will be used to parallelize the execution of the applications supporting concurrent code execution, automatically detecting and enforcing the data dependencies among components and spawning parallel tasks to the available resources, which can be nodes in an edge cluster or clouds. COMPSs runtime will also support the serverless computing paradigm, and application components or COMPSs workers will be run as event-triggered functions (for inference in the edge for example) orchestrated within a workflow. To this aim, the SCAR framework will be used to detect the availability of fresh data on an object storage and to deploy the required containers with COMPSs functions.

For the scheduling of training tasks on accelerated devices, the rCUDA will be used to manage the efficient allocation and sharing of resources minimizing the training costs in terms of energy for the private data center or public cloud operating cost.

The optimal solution identified by the design time tool will be periodically reevaluated in order to account for load variations. SPACE4AI-R will evaluate the current status of the system, in terms of incoming loads and response times of the different components. According to this, it will provide an updated solution in terms

[10] https://github.com/deephealthproject/eddl

of components configuration and their deployment on the available resources, optimizing components response times and execution costs. Application components migration will leverage Krake and be supervised by SPACE4AI-R when performance constraints are introduced, in order to provide better Service Level Objectives. In the same way, PyCOMPSs runtime will leverage SPACE4AI-R if performance constraints are introduced, otherwise COMPSs tasks will be managed by the COMPSs default scheduler.

Both the application components and the core elements of the AI-SPRINT runtime will be monitored through the AI-SPRINT Monitoring Infrastructure. The alerting module (implemented as an InfluxDB component) will notify of QoS thresholds violations Krake, SPACE4AI-R or COMPSs runtime, according to the application and AI-SPRINT framework deployments.

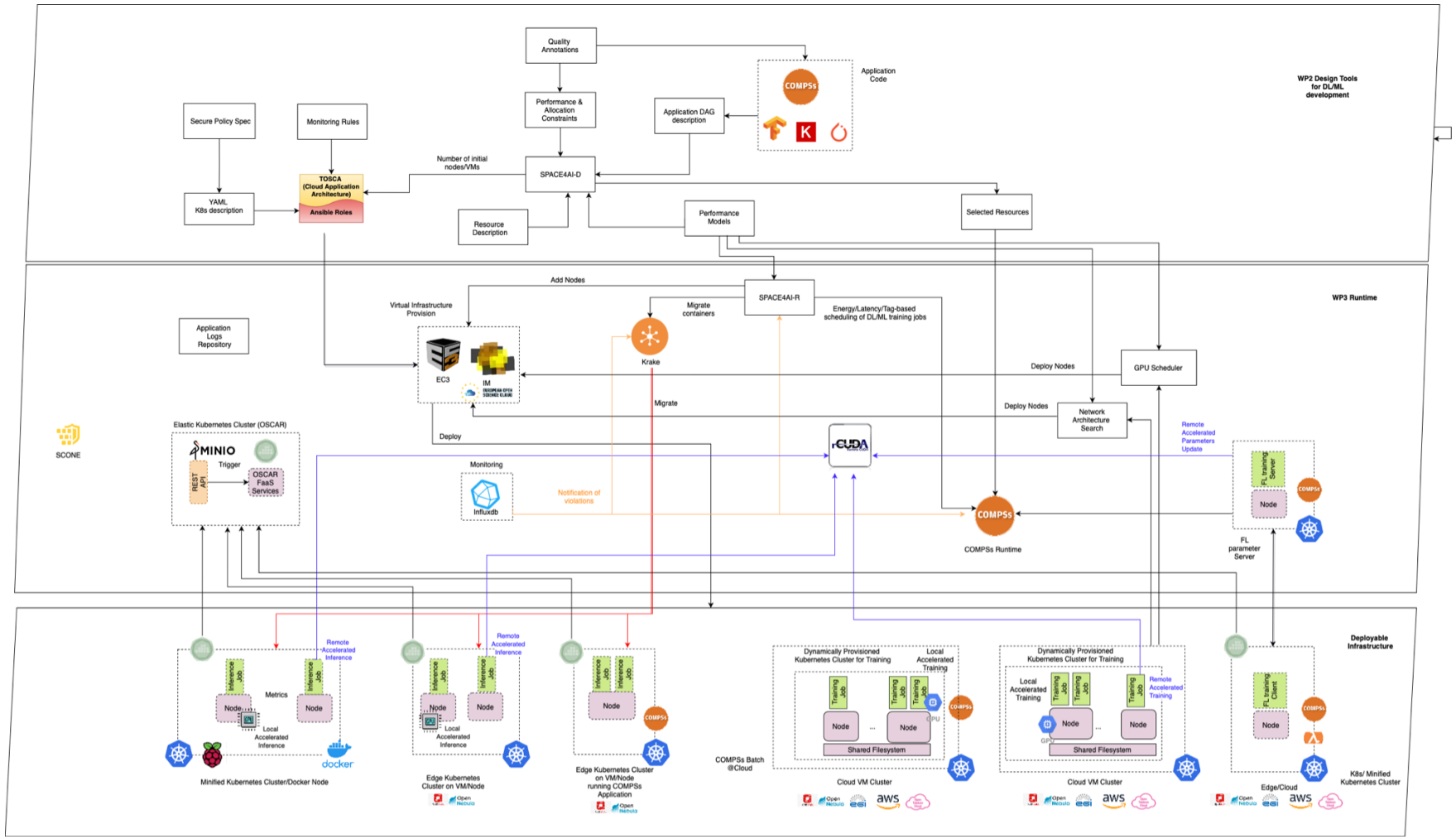


Figure 4.1 - AI-SPRINT Detailed Architecture

4.2 Integration plan

The integration plan aims at harmonizing the definition of the requirements with the design and development phase and ensuring that scientific and technical activities comply with the use cases definition. In AI-SPRINT a specific task in WP1 is devoted to these integration activities and has also a design side, though it is confined at the level of architectural design, as described in this document.

The design and implementation of the AI-SPRINT framework are performed by Work Packages WP2, WP3 and WP4. These WPs develop the tools that will be proposed to WP5 (within the project) and to the world. These WPs have the goal of translating WP1 requirements into tools. While executing such activity, they also provide valuable feedback to WP1, which enables more precise alignment of the outcome of AI-SPRINT to real-world needs; also, such feedback acts as a verification of WP1 results.

The integration plan is validated through verification activities and milestones in the workplan of the project, as depicted in Figure 4.2; the first validation corresponds to WP5 and its goal is validating the output of WP2-WP4 by using such output to build actual applications. More precisely, the objectives of WP5 are to validate: i) the output of WP2 by using it to create AI-based applications (such applications intentionally belonging to widely heterogeneous categories, in order to stress the AI-SPRINT framework from different perspectives); ii) the output of WP3 by using its tools to deploy, execute, monitor, and operate AI applications preserving the privacy of data; iii) the output of WP4 by verifying that the structure and execution of the AI applications (and especially of the one concerning healthcare) meet the security requirements set forth by WP4 and incorporated first in the requirements generated by WP1, and later into the tools produced by WP2 and WP3.

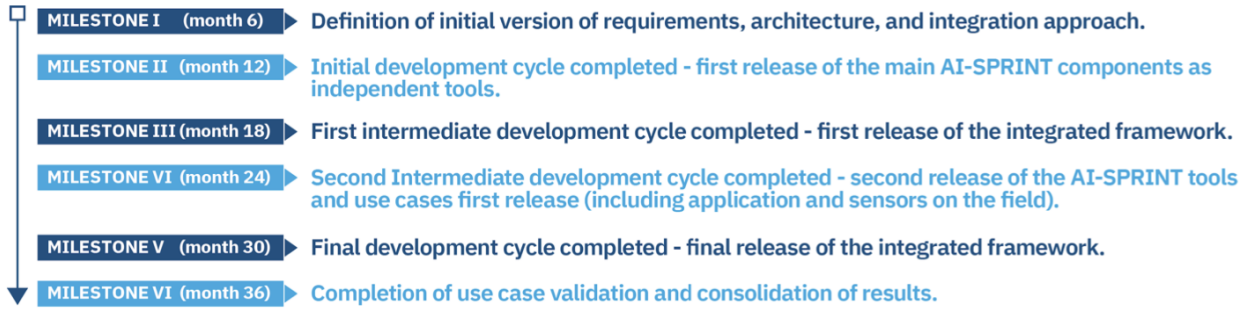


Figure 4.2 - Milestones for components development

A detailed roadmap for each set of components is available in Sections 6.2, 6.3 and 6.4 of the *AI-SPRINT Deliverable D1.2 - Requirements Analysis document.*

5. Conclusions

This deliverable has provided the initial version of the architecture of the AI-SPRINT platform. This document is one of the results of the first six months of activities that included the collection of the requirements from the use cases (detailed in the Deliverable D1.2) and the selection of the technologies that compose the basis of the platform, that have been compared to the current state of the art in the field (deliverable D1.1). These activities have involved partners from multiple work packages (WP1, WP2, WP3, WP4, WP5).

This document has presented the different layers of the AI-SPRINT architecture that include the Design Tools, the Runtime Framework, the Deployable Infrastructure and the Security stack. For each of these layers the document provides details on the interactions between the components, and it provides a way, through sequence diagrams, on how to implement the required use cases.

The final version of this document will be provided at M24, after completing the different phases of developments that will include the feedback of the use cases.

6. Appendix A

A.1 Roles in AI Design and Operations

During the requirements elicitation exercise (Task 1.2 - Requirements Analysis) a number of end-users (personas) for the AI-SPRINT framework have been identified. More details are available in the *AI-SPRINT Deliverable D1.2 - Requirements Analysis*, Section3. Here we summarize the relevant roles that will be involved in the definition of the interactions of the AI-SPRINT assets.

Role	Responsibilities
Application End User	The final end-user an application
Application Architect	Define and review the overall architecture of the application Define KPIs (AI and application) Review performance and propose improvements Define and coordinate development tasks
Application Developer	Execute Development tasks Develop and Enhance the software and deploy new versions
AI Expert	ML/AI model design and implementation
Application Manager	Application Management: Monitoring, Change management Owner of the CD process
Infrastructure Provider & Sysops	Setup and Management of the infrastructure runtime environment
Application/Service Provider	Service Delivery Management of the AI-SPRINT Solutions: Operational Performance, Cost management, SLA management

Table 2. 1 - AI-SPRINT end-users

A.2 Design and Programming Abstractions

Summary of the use case

ID	DES-UC1
Title	AI applications programming
Priority	Must have
Actors	Application Developer, AI Expert
Pre-conditions	The application developer provides his/her sequential code
Post-conditions	The code is enriched with annotations to provide hints for the scheduling and for the design space exploration.

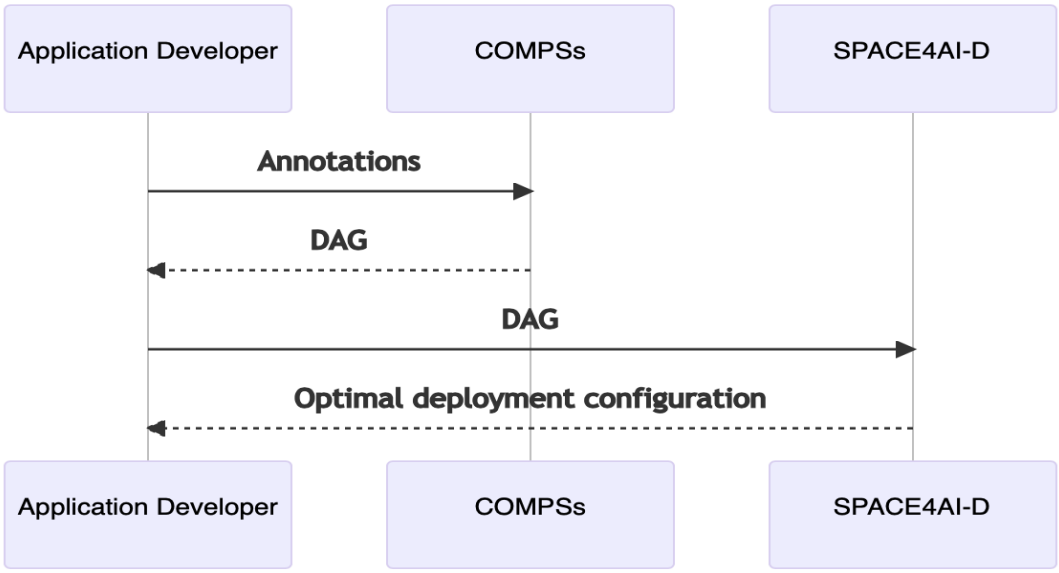
Description of interactions

The definition of the application components in AI-SPRINT is based on the PyCOMPSs programming model which follows a sequential paradigm where the application is a plain Python script whose functions are annotated by the user; these annotations are used by the runtime to run those parts of code as asynchronous parallel tasks. The decorators also contain a description of the function parameters, such as type and direction, etc. which is the basic information for building the dependency graph where tasks are represented as nodes and data dependencies between tasks as edges.

Another important feature of PyCOMPSs is the capability to express workflow dynamicity. PyCOMPSs workflows are created at execution time from a Python code, so it is very easy to program different workflow branches from previously generated values. However, dynamicity can be also generated by failures or exceptions. Scientific workflows usually implement hyperparameter searches in huge spaces performing loads of simulations with different input parameters. It is very likely that some of these simulations fail, but they should not imply a failure in the whole workflow. For those simulation tasks, developers can provide hints to ignore these failures, or to cancel their successors. It is also possible that a solution is found before finishing all the execution. For this purpose, PyCOMPSs supports parallel try-except blocks where if a specific exception is raised in one of the tasks of the block all the remaining tasks will be cancelled. In this way, applications made of components deployed on unstable devices in the edge can be executed successfully thanks to these fault-tolerance mechanisms.

The annotated code together with the execution graph generated by PyCOMPSs runtime, are used by the SPACE4AI-D tool to provide the type and the number of resources that minimize the cost and the execution time.

Sequence diagrams



Data flows

Data is provided by the user as annotations in the code to describe the type of parameters and constraints on the resources. This information is processed by the PyCOMPSs runtime to perform a matchmaking with

the available resources provided as initial set according to SPACE4AI-D. The graph of the execution is then sent to the Application Design Space Exploration tool that returns the optimal deployment configuration of the application components.

A.3 Performance Models

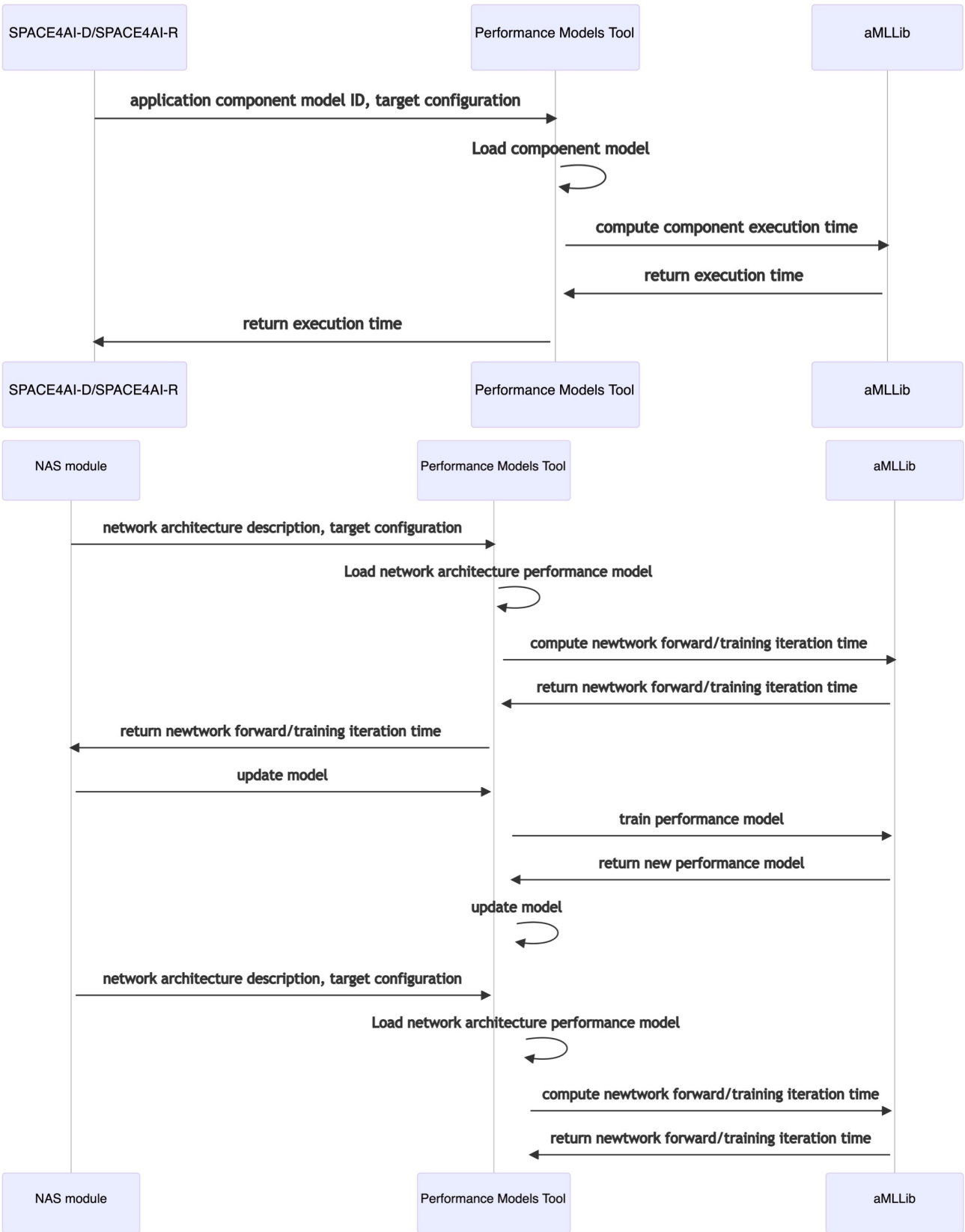
Summary of the use case

ID	DES-UC2
Title	Performance Models Tools
Priority	Must have
Actors	Network Architecture search module, SPACE4AI-D, SPACE4AI-R, GPU scheduler
Pre-conditions	Application profiling has been performed and a model is built through the aMLLib library
Post-conditions	Performance models tools return the estimated execution time of an application component according to the selected configuration

Description of interactions

The external tool which needs the execution time estimate (SPACE4AI-D/SPACE4AI-R and the Network architecture search module in the diagram illustrated below) provides the target configuration description and invokes the performance models tool. Then the performance models tool, by relying on the aMLLib library (https://github.com/eubr-atmosphere/a-MLLibrary), computes the execution time. The interaction between the performance models tool and the Network architecture search (NAS) component is more complex. Indeed, the NAS module in its exploration performs a profiling of the target network architecture. In this way, the NAS module enriches the training set used by the aMLLib library to re-train the performance model regressor that is used under the hood to compute the prediction.

Sequence diagrams



Data flows

The data provided to the performance model tool include the component regression model ID and the resource target of the deployment description. The performance model tool returns the estimate of the component execution time in seconds. In the interaction with the NAS module, periodically a new training set for the aMLLib is provided and the performance model is retrained.

A.4 AI Models Architecture Search

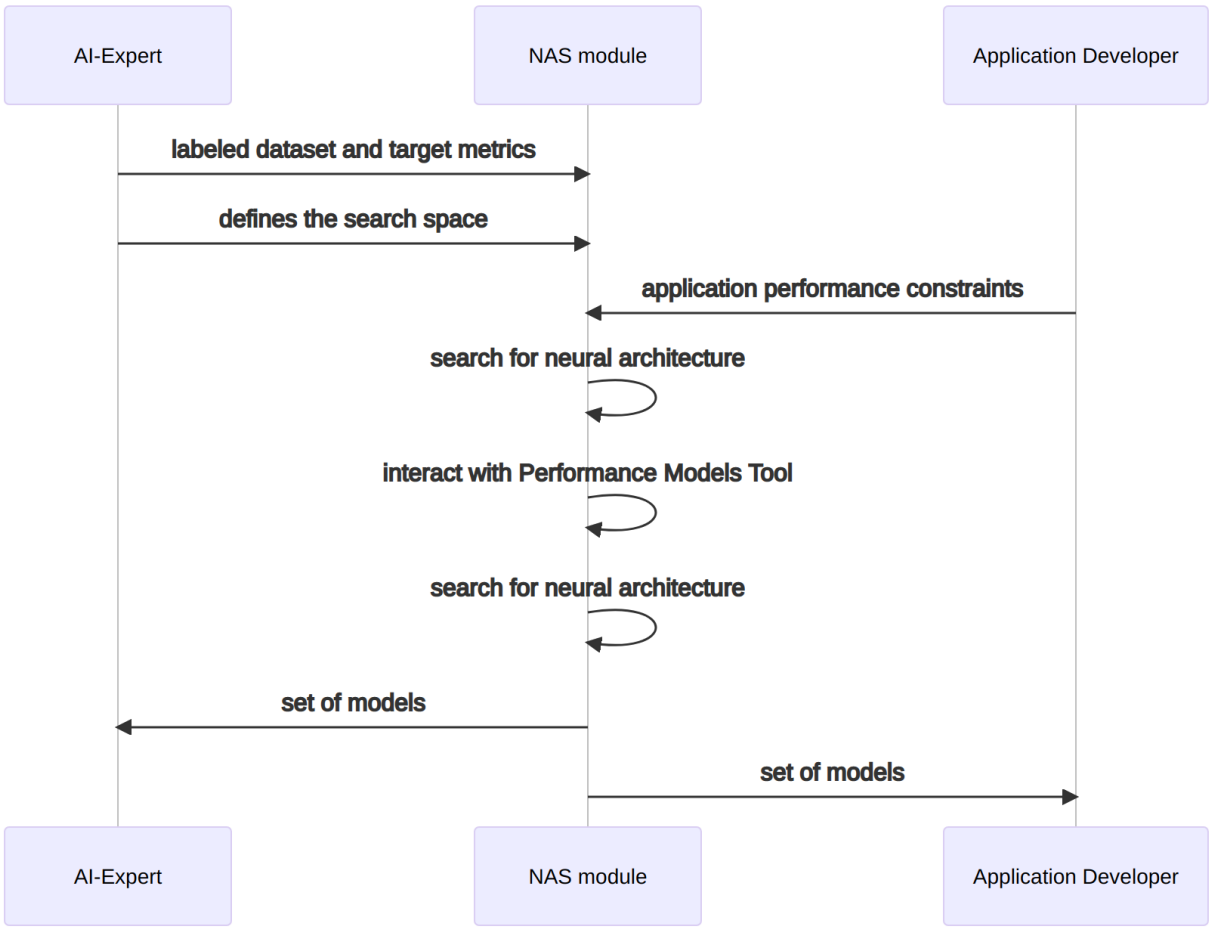
Summary of the use case

ID	DES-UC3
Title	Neural Architecture Search
Priority	Must have
Actors	Application Designer, AI Expert
Pre-conditions	Data have been collected and labeled or a suitable loss function has been defined regarding the data. Requirements for the final deployment of the model have been defined and formalized.
Post-conditions	The neural architecture search tool returns a set of possible architectures which fulfill the required task and under the constraints defined by the AI Expert and the Application Designer.

Description of interactions

The Neural Architecture Search tool interacts with the AI Expert which will provide the required dataset labeled for the specific application and suitable metrics to be optimized by the search tool. The AI-Expert will also define the search space for the Neural Architecture Search, possibly limiting the search space herself. The Application Developer will define the set of performance constraints for the final model implied by the application, e.g., latency, MFlops, memory, power consumption, privacy requirements. The NAS, during the search of the optimal architecture will interact with the Performance Models Tool asking for performance predictions so to shorten search time, retraining periodically the performance models according to the current cells structures (see also the use case in Section A.2).

Sequence diagrams



Data flows

The data provided to the NAS are a dataset and the loss definition as well as the definition of the search space via CLI parameters. Also the resource target for the deployment and the performance constraints are given. The NAS module returns a fully trained model to be deployed as part of the application.

A.5 Application Design Space Exploration

Summary of the use case

ID	DES-UC4
Title	Application Design Space Exploration
Priority	Must have

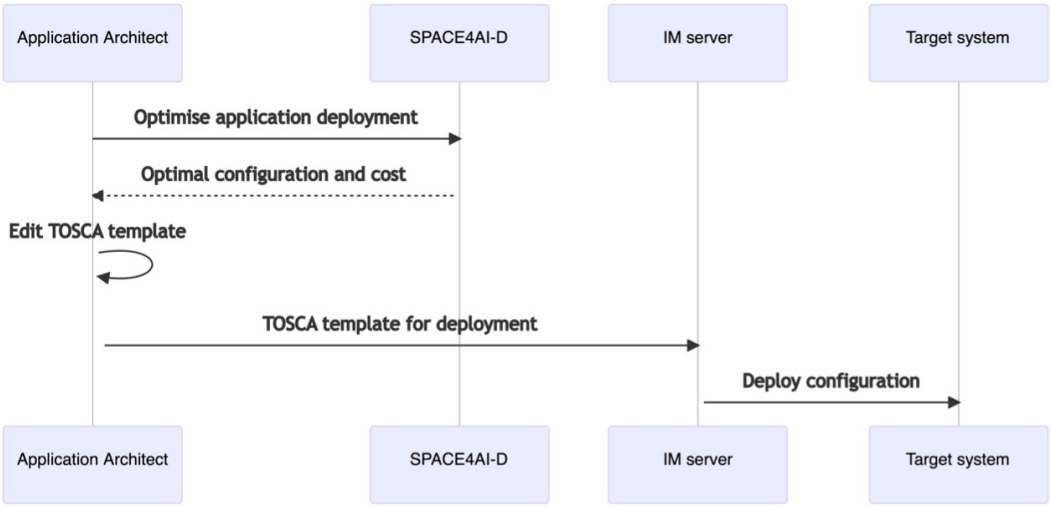
Actors	Application Architect
Pre-conditions	The Application Architect submits a description of the system, including application components and its memory requirements, resources (located in cloud and edge) and their available memory (which bounds the maximum number of components that can be co-located in each device) and costs. The application is modeled as a direct acyclic graph (DAG) and it is associated with the expected workload (e.g., number of requests per seconds). Each component is associated with candidate resources for its execution and the corresponding performance demanding time. Moreover, local and global performance constraints are specified: Local constraints predicate on a single component (e.g., exposure check for a picture has to be lower than 100ms) while global constraints involve a set of components that are running one after another and are associated with a global threshold.
Post-conditions	SPACE4AI-D explores and finds an optimal solution for the application component placement problem (according to candidate resources introduced by the AI developer) minimizing the resource cost and satisfying the performance constraints. The optimal configuration includes a description that shows the placement of each component, provides the estimated execution time of each component and the total cost of the system.

Description of interactions

Application design space exploration is a process to explore and find an optimal solution for the application component placement. The system information is submitted by the Application Architect and includes application components, candidate system resources, the application model (as a direct acyclic graph - DAG), compatibility assignment among components and resources, resource costs, component performance demands, local and global constraints. Application components are organized in computational layers which include a set of candidate resources that can be selected and which allow to specify the characteristics of the networks that connect edge and cloud.

SPACE4AI-D, explores and finds the optimal solution for the application component placement problem minimizing the resource cost and satisfying the performance constraints. *SPACE4AI-D* returns the optimal configuration and resource cost to the Application Architect who then edits the TOSCA template according to the optimal configuration found. The Application Architect then sends the TOSCA template to the Infrastructure Manager (IM) server which finally deploys the application on the target system.

Sequence diagrams



Data flows

The main data items consist of the description of the system including components' name and their memory requirements, computational layers and the resources located on them. Resources are characterized by some properties like name, type (edge or cloud), memory, the number of available instances and cost. Moreover, the SPACE4AI-D description file associates each component with the compatible resources which can support its execution and introduces components performance demands. Moreover, the description file also includes the list of local and global constraints and DAG application model, associating each node of the DAG (representing a component) with the transition probabilities and data transfer requirements. The optimal configuration consists of the description of the optimal placement which shows (in the same format of the input description file) the resources allocated to each component. The last element involved in this scenario is the TOSCA template which is edited by the Application Architect according to the optimal configuration found.

A.6 Deployment tools

The requirements analysis from the use cases described in AI-SPRINT *Deliverable D1.2 - Requirements Analysis* results in the following scenarios regarding the automated deployment of virtual infrastructure.

Summary of the use case

ID	RUN-UC1
Title	Deployment tools
Priority	Must have
Actors	Application Manager, Infrastructure Provider
Pre-conditions	An end user submits a deployment request to provision an application / virtual infrastructure from an IaaS Cloud provider/private cloud. The deployment request is specified as a TOSCA template directly to the IM server. Alternatively, the user

	selects the application / virtual infrastructure from the IM dashboard, which builds the TOSCA template and delegates it to the IM server.
Post-conditions	The virtual infrastructure together with the application running inside will be automatically provisioned from the IaaS Cloud/private cloud. The user will receive access credentials to that infrastructure or the endpoint to the application. The user will be able to manage the lifecycle of the virtual infrastructure.

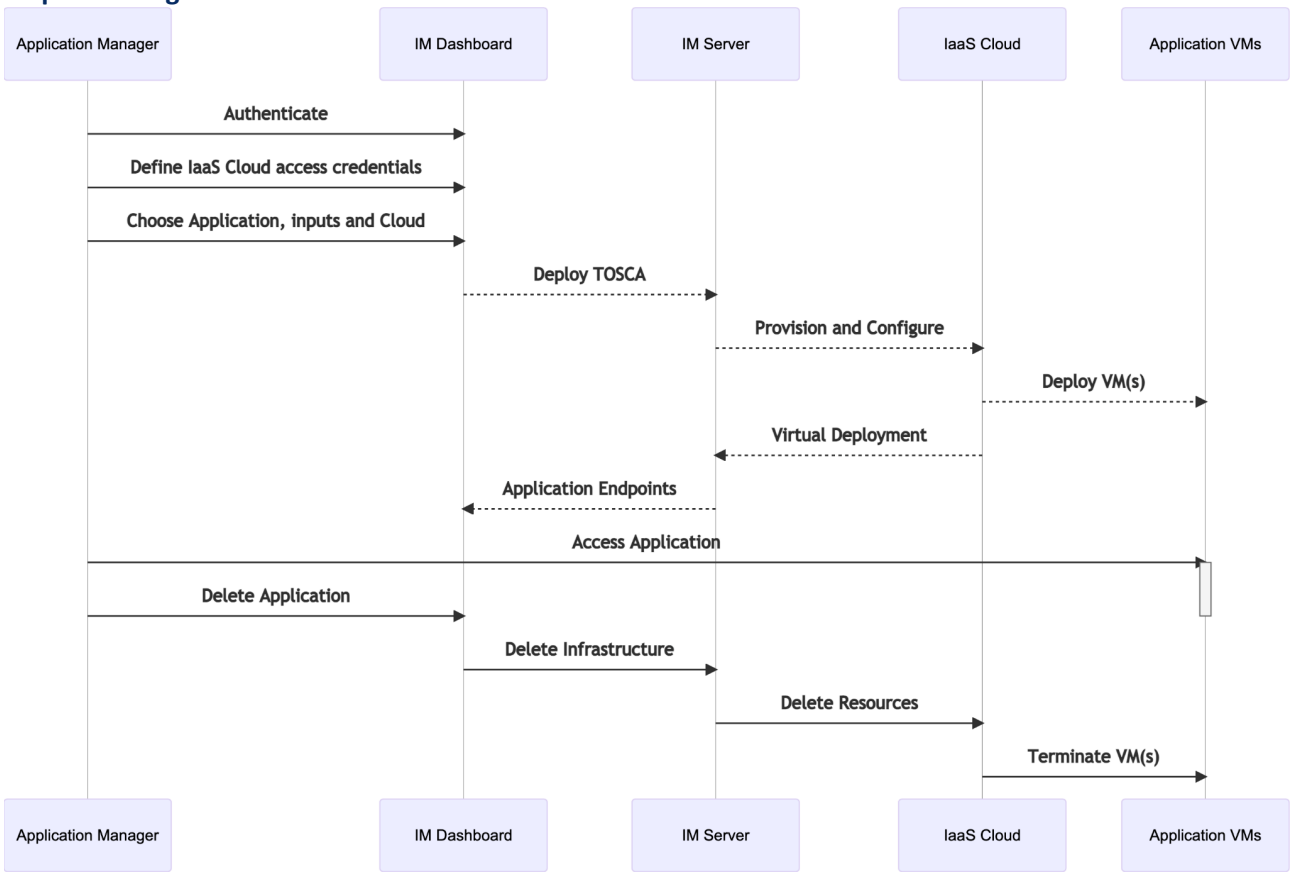
Description of interactions

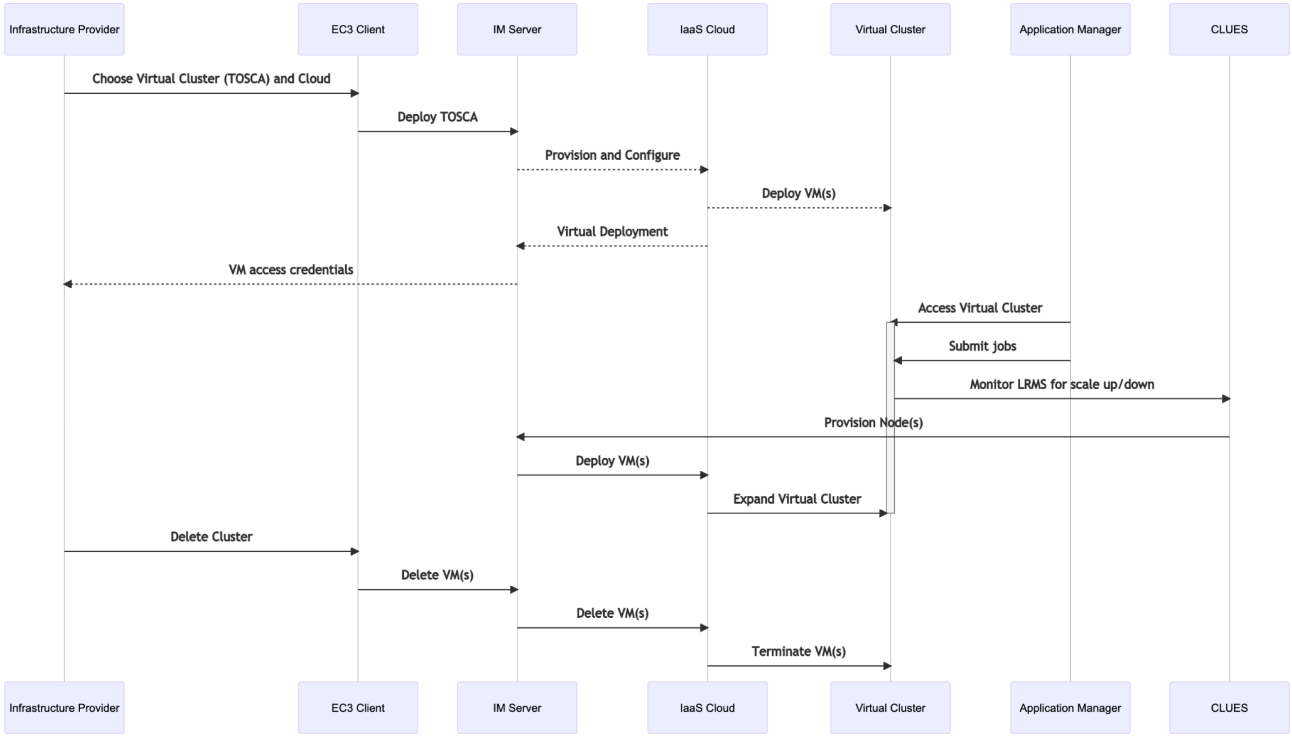
Provisioning virtual infrastructure from several IaaS Clouds requires interacting with their different APIs and introducing automated means to perform configuration and application deployment. The deployment tools involve mainly three components: the Infrastructure Manager (IM) server, which performs the automated deployment and configuration of virtual infrastructures supporting the TOSCA specification; the IM dashboard, which provides a simplified web-based GUI for the end users to select predefined application recipes that can be partially configured by the end-user before its deployment on the IM and, optionally, EC3 a client-side tool for the IM that focuses on the deployment of elastic virtual clusters that can grow and shrink depending on the workload.

The first sequence diagram illustrates how a user performs the deployment of an application and virtual infrastructure on an IaaS Cloud. First, the user authenticates with the IM dashboard and defines the IaaS Cloud access credentials of all the Cloud providers that will be used. Second, the user chooses the application, provides input information in case of parameterised TOSCA templates and selects the IaaS Cloud on which the application and the virtual infrastructure will be deployed. This results in a TOSCA template that is built and submitted to the IM server. Programmatic access via the IM server's REST API or via the IM CLI will start from this step. The IM server performs a syntactic analysis and contacts the IaaS Cloud's native API in order to provision and configure the VMs (Virtual Machines) that will support the execution of the application. Once deployed, depending on the configuration of the TOSCA template the IM dashboard will show the application's endpoint to be accessed by the end user or provide access details to the underlying VMs through SSH. The user will be able to manage the lifecycle of the virtual infrastructure by adding and removing nodes (not shown in the diagram because the infrastructure may feature automated elasticity). Finally, the user triggers the termination of the infrastructure which results in releasing the required resources.

The second sequence diagram describes the deployment of an elastic virtual cluster using EC3 using CLUES as the elasticity manager to decide when to add or remove additional working nodes of the cluster depending on the workload. First, the user employs the EC3 client to submit a TOSCA template to the IM server, in charge of provisioning the required VMs from the IaaS Cloud, and configures them as a cluster. Several types of clusters are supported such as Kubernetes or SLURM. The user receives access credentials to the cluster front-end node via SSH, from which the user can submit batch processing jobs. The CLUES elasticity manager monitors the number of pending jobs at the LRMS (Local Resource Management System) in order to decide when to scale out and scale in, in terms of number of nodes and according to a set of pre-configured elasticity rules. CLUES contacts the IM server to provision additional nodes (typically a specific instance of the IM server installed within the front-end node of the cluster so that the cluster is self-managed). These nodes are automatically integrated in the LRMS to cope with the increased number of pending jobs. Once the LRMS is empty the nodes are terminated (not shown in the diagram). Finally, the user can terminate the cluster resulting in the elimination of the resources provisioned.

Sequence diagrams





Data flows

The main data flow consists in the description of the application architecture by means of a TOSCA template. Also, user credentials to the underlying IaaS Cloud are required to perform the provision of Virtual Machines. VM identifiers are received by the IM and employed in order to manage the lifecycle of virtual infrastructure by the user.

A.7 Monitoring tools

The requirements analysis from the use case results in the following scenarios regarding the monitoring of the cluster and deployed applications.

Summary of the use case

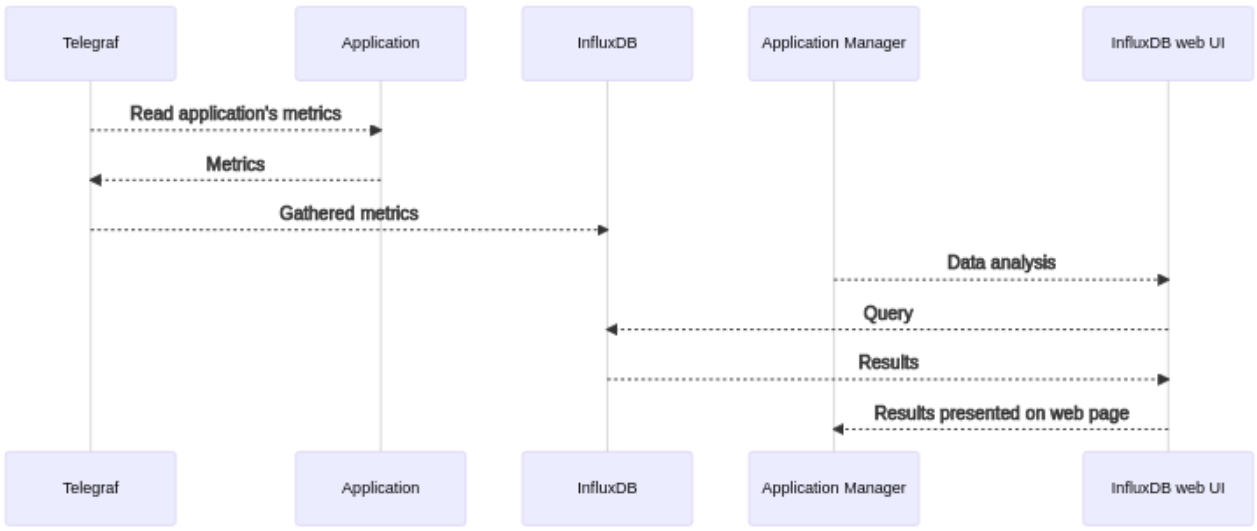
ID	RUN-UC2
Title	Monitoring of the running applications
Priority	High
Actors	Application Manager, Infrastructure Provider & Sysops
Pre-conditions	Applications are provisioned and running in the infrastructure.
Post-conditions	Application metrics stored in the monitoring infrastructure. Subsystems responsible for application management informed about spotted anomalies. Application Manager & Sysops can view system's statistics and collected metrics in the dedicated web UI. Appropriate subsystems can query and analyse stored data.

Description of interactions

The monitoring infrastructure will consist of: (i) a central InfluxDB cluster (plus additional tools) responsible for storing and analysing data, and (ii) various components (SDK libraries or Telegraf) responsible for gathering data, deployed with monitored applications. These components will use data provided by applications and their environments, to generate data entries containing metrics. Metrics are then sent to InfluxDB. The general rule is that an application should not "know" that it is monitored by this specific infrastructure. It should only provide metrics, using - usually - mechanisms provided by standard application frameworks or common libraries. This is a typical "pull" scenario, when monitoring decides when to read data from monitored applications. In such a case data gathering is performed periodically. The other (rare) "push" scenario is implemented when metrics generated by application must not be lost and are expected to be stored in the monitoring data storage. In this case, the application sends data using a dedicated SDK. In such a scenario the application also has to provide necessary metadata so the monitoring subsystem properly classifies and stores data.

In the InfluxDB, one can define special rules which can be checked using gathered time-series data. External systems can be notified when a rule has been violated.

Sequence diagrams



Data flows

Metrics consist of field values (usually numbers, but they might also be short texts) and additional tags (labels, key-value pairs). This data record can be interpreted as a value describing the state of a specific instance of the application. Provided tags help distinguish among data collected from many, different instances of the same application. In case of a “pull” scenario, data is fetched by a monitoring component responsible for data gathering on the client side, and then “enriched” with a timestamp. Then it is stored in the data queue and then sent to InfluxDB using the appropriate API. In the case of “push” scenario, the application decides when to send metrics. The SDK is responsible for “enriching” the data with timestamp and sending collected data to the monitoring system.

InfluxDB receives and stores data. Data stored in the database can be queried and analysed using the Flux query language.

A.8 Programming framework runtime

Summary of the use case

ID	RUN-UC3
Title	Training of models
Priority	Must have
Actors	AI Expert
Pre-conditions	The AI Expert submits the training of a model, developed using the Design Time Tools, in batch mode on a cloud cluster

Post-conditions	Application's components are deployed on the target platform and resources assigned to the runtime for execution. The model is trained and saved in a storage.

ID	RUN-UC4
Title	Inference of pre-trained models
Priority	Must have
Actors	AI Expert
Pre-conditions	The availability of new data coming from the field triggers the execution of inference functions on an existing trained model.
Post-conditions	Application's components are deployed on the target platform and resources assigned to the runtime for execution. Output data of the inference process are stored back in the storage.

Description of interactions

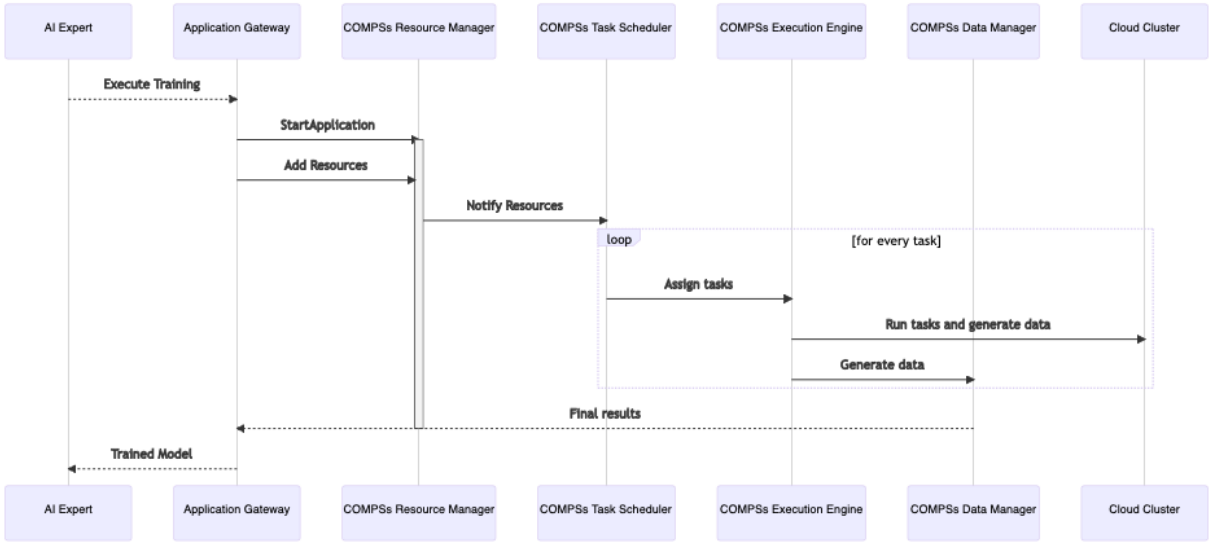
The Programming Framework Runtime, based on PyCOMPSs, offers an interface where to request the execution of functions that will run either on the computing resources embedded on the host device or transparently offloaded onto other nodes on the cloud-edge continuum. The entry-point to the agent is its API which offers methods for requesting the execution of a function with a certain parameter value and performing resource pool modifications. Users or applications request a function execution detailing the logic to execute, the resource requirements to run the task, the dependencies with previously submitted tasks and the sources where to fetch the data involved in the operation. Upon the reception of a request, the API directly invokes the COMPSs runtime system. The goal of this runtime is to handle the asynchronous execution of tasks on a pool of resources. To achieve its purpose, the runtime has four main components. The first one, the Resource Manager (RM), keeps track of the computing resources – either embedded on the device or on remote nodes– currently available. Upon the detection of a change in the resource pool, the RM notifies all the other components so they react to the change. If dynamic resource provisioning is enabled, this component should also dynamically adapt the reserved resources to the current workload. The second component is the Task Scheduler (TS). Its purpose is to pick the resources and time lapse to host the execution of each task while meeting dependencies among them and guaranteeing the exclusivity of the assigned resources. The scheduling policy followed by the TS is selected at start time, and can be replaced by new policies since they are implemented in a plug-in fashion. The Data Manager (DM), the third piece of the runtime, establishes a data sharing mechanism across the whole infrastructure to fetch the necessary input data to run the task locally and publish the results. The last component, the Execution Engine (EE), handles the execution of tasks on the resources. When the TS decides to offload a task to a remote agent, the EE forwards the function execution request to the API of the remote agent. Conversely, if the TS determines that the local computing devices will host the execution of a task, the EE fetches from the DM all the necessary data missing in the node, launches the execution according to its description and the assigned resources, and publishes the task results to the DM. It is during the task execution when the

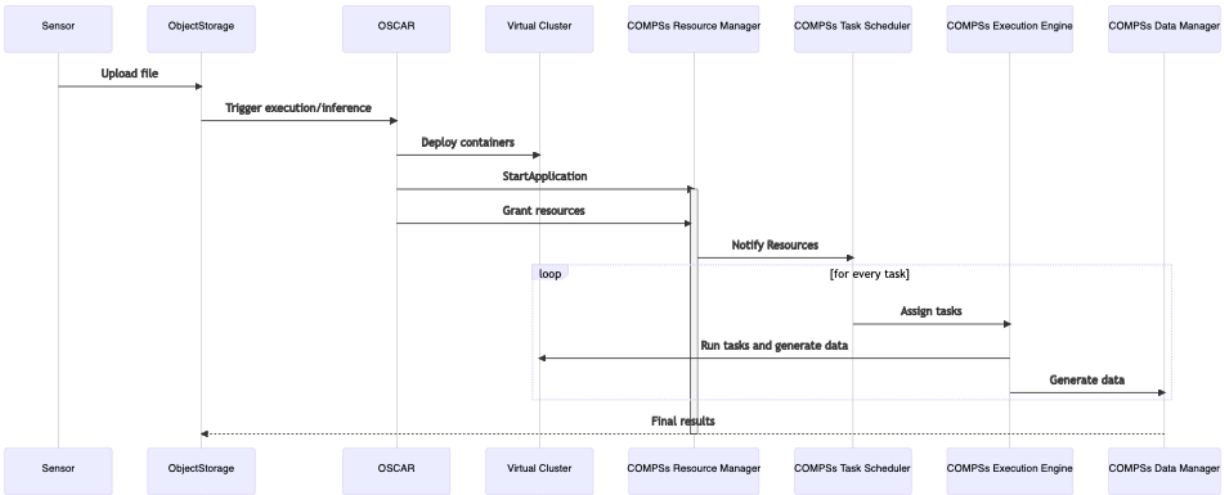
task-based programming models take the scene and convert the logic of the method into a workflow composed of tasks whose execution will be requested to the same agent.

The first sequence diagram illustrates the execution of a training process initiated by an AI Expert. The application components, annotated using the design tools, are deployed on the target platform on a set of virtual nodes and the PyCOMPSs runtime takes care of their orchestration and parallelization. The results of this process is a trained model that can be exported and used for inference phases.

The second sequence diagram depicts the triggering of inference functions due to the availability of new data to feed the model. In this case the OSCAR framework is responsible for the deployment of the containers on a virtual cluster in the edge. The PyCOMPSs runtime is activated, the model is loaded and the proper functions are invoked in a FaaS style. Output data is made available in the object storage and resources are released.

Sequence diagrams





Data flows

In the case of the model training the data flows consist in the input data of the model and in the output trained model available in a standard format. For the inference case, data is generated in the edge by a sensor or instrument and provided as input of a trained model. Output data is saved back in the storage.

A.9 Federated Learning

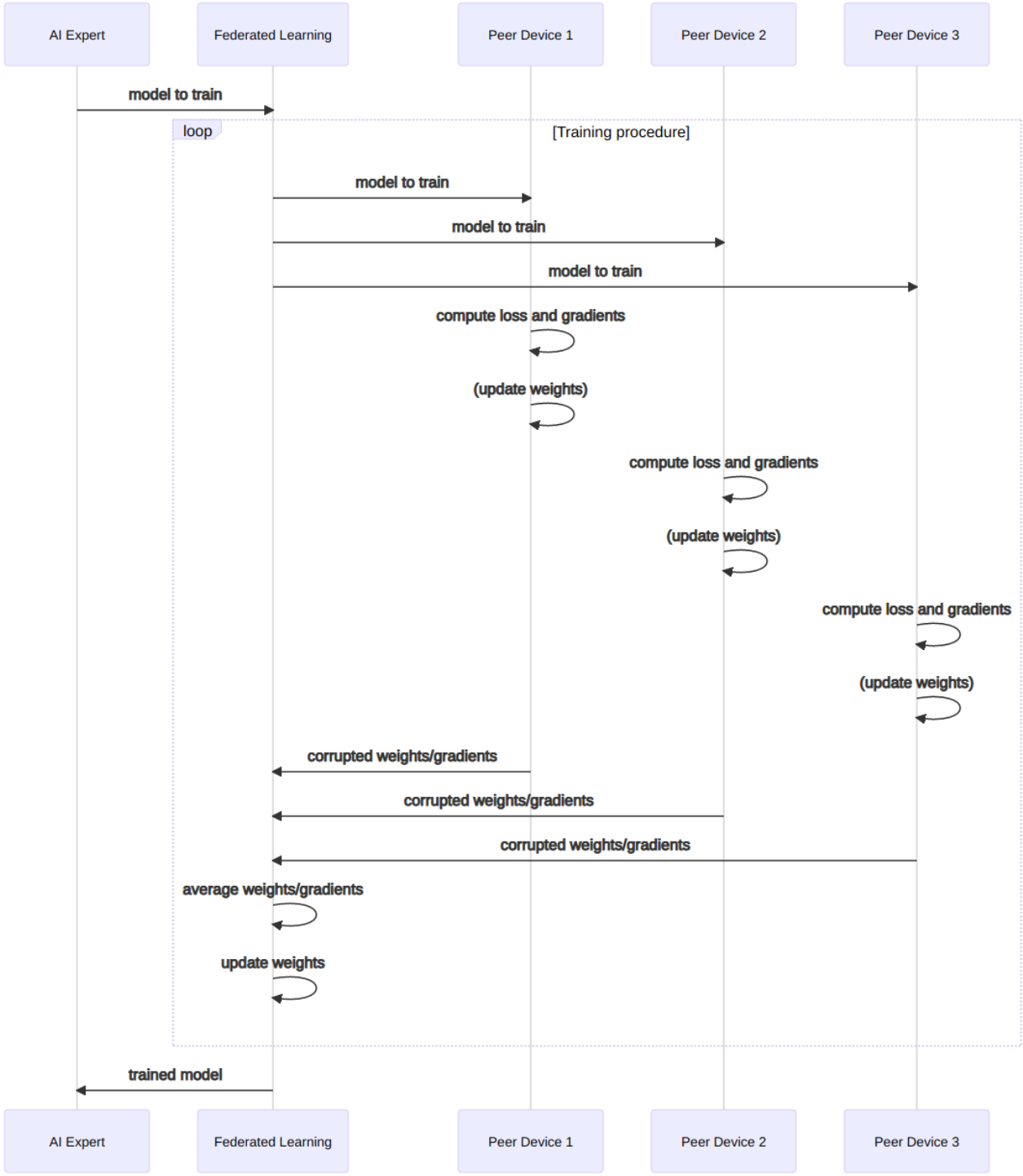
Summary of the use case

ID	RUN-UC5
Title	Federated Learning
Priority	Must have
Actors	AI Expert
Pre-conditions	A model has been defined to be trained either via expert knowledge or via the NAS module applied to a subset of data. A set of peers devices has been selected for training, including their datasets and the loss function.
Post-conditions	A centralized model has been trained leveraging the dataset of the pears, no data has been exchanged among the peers.

Description of interactions

The AI Expert defines the models to be trained based on his previous knowledge and some data, possibly simulated. In this initial architecture definition she can also use the Neural Architecture Search model on the data she has for the design of the architecture. Once a model to be trained is defined this is sento to the Federated Learning algorithm which stores the master model centrally and sends a copy to the peers in order for them to train their own copy with their own data. To do such, each peer computes the local error with respect to its dataset and the corresponding gradients. In case the implementation uses model averaging the model is updated and the weights are sent to the centralized model for averaging. otherwise only gradients are sent instead of the full model. To preserve privacy, mode weights or gradients are corrupted with samples from a zero mean so that the overall gradient can be recovered via weight/gradient averaging, but model inversion for each of the local models is not possible.

Sequence diagrams



Data flows

Architectural description of the model to train is sent to the Federated Learning module, then the Federated Learning module sends a copy of the centralized model to each of the peers. The peers send to the Federated Learning model either their local models weights or the computed gradients. The Federated learning model returns a fully trained model to the AI Expert.

A.10 Scheduling for accelerated devices

Summary of the use case

The requirements elicitation of D1.2 considers led to the following use case for the scheduling for accelerated devices components.

ID	RUN-UC6
Title	Scheduling for accelerated devices
Priority	Must have
Actors	AI Expert
Pre-conditions	The AI expert submits one or more training jobs to an in-house private GPU based cluster or to a public cloud. Training jobs are provided as Docker containers.
Post-conditions	Jobs are executed minimising operation costs. If the cluster is idle for more than 15 minutes it will be deallocated. In case of disaggregated hardware resource configurations, remote GPUs are accessed through rCUDA

Description of interactions

The scheduling of training jobs is a result of a sophisticated process, which is carried out by the Scheduling for accelerated devices component. The component includes three main modules: the Job Manager, the Job Profiler, and the Job Optimizer. The Scheduling for accelerated devices relies on rCUDA when disaggregated hardware resource configurations are considered. Training applications are submitted by AI experts in the form of Docker images. The Job Manager receives the users' requests and orchestrates the execution of the submitted jobs as follows. In case of private deployments, if the job has never been executed before, the Job Profiler is invoked to collect information about the expected per-epoch execution time of the submitted job on the different resources available in the system. One free node (characterized by a given type and number of GPUs) is dedicated to collect these data, which are then stored in a database (in case no idle nodes are available, the job profiling is postponed until one among the running jobs will end). Per-epoch time will also be updated after the training job finishes. If the cluster is heterogeneous (i.e., it includes nodes with different CPU or GPU types), multiple nodes can be engaged in profiling activities. The collected profiling data, together with a description of the system, with all available nodes, are provided to the Job Optimizer, which is in charge of selecting the optimal deployment for the submitted jobs. In particular, the Job Optimizer determines which resource configuration should be used to deploy and run the different jobs. If the resources available in the system are insufficient to run all jobs concurrently, some of them are preempted, and their execution is postponed. The Job Optimizer is invoked by the Job Manager periodically, or in reaction to re-scheduling events (i.e., a new job submission or completion). The Job Manager deploys the jobs according to the optimization results, migrating the ones in progress among different GPUs if necessary.

In case of public cloud deployments, if no VMs are already running at the time of the job submission, the VM identified by the Job Optimizer will be deployed through IM. If the cluster is idle for more than 15 minutes it will be deallocated.

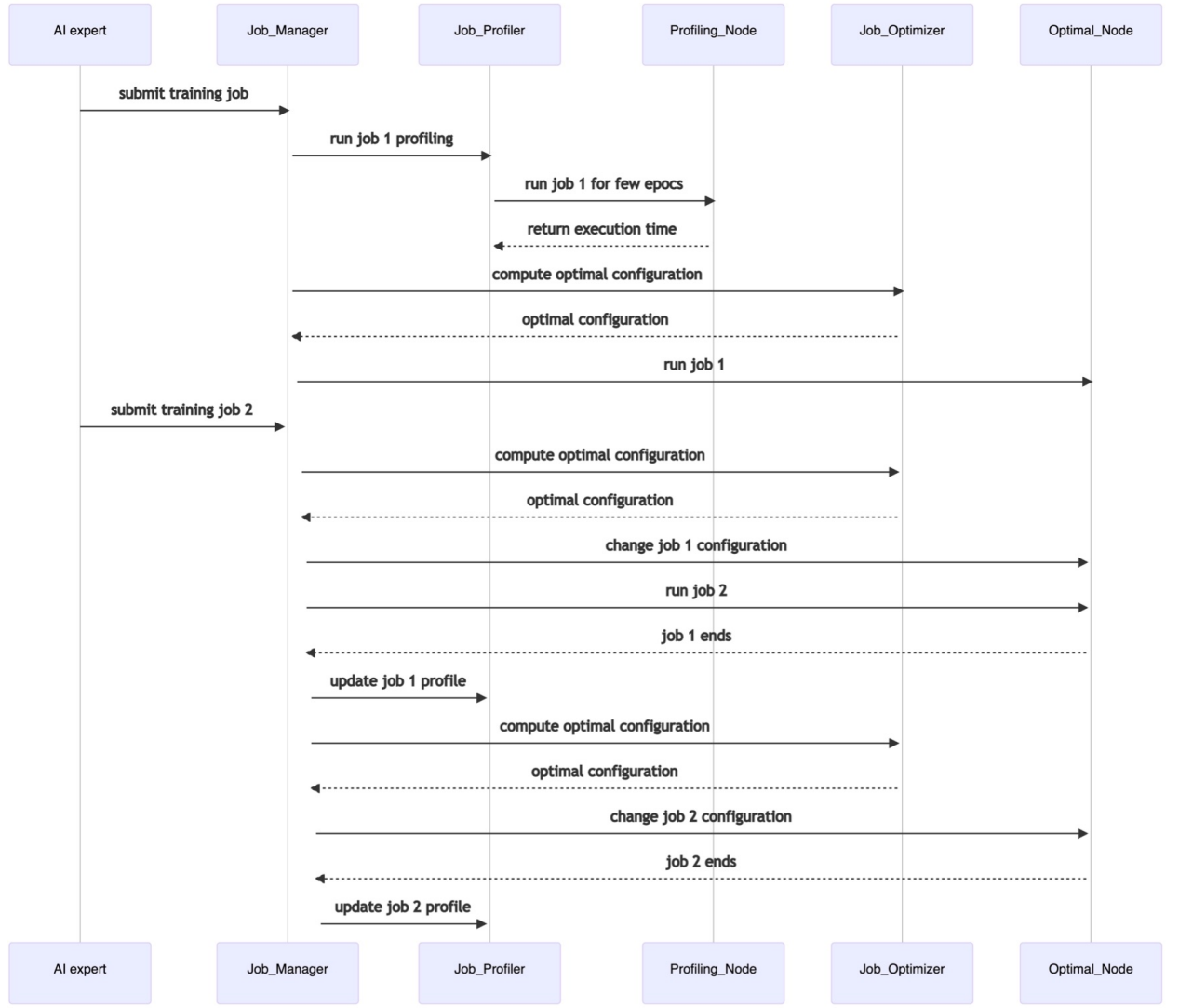
The first sequence diagram illustrates how an AI expert puts the process in motion in a private cloud. The end user initially submits job1 which was never run on the system before. Then the Job Manager sends a

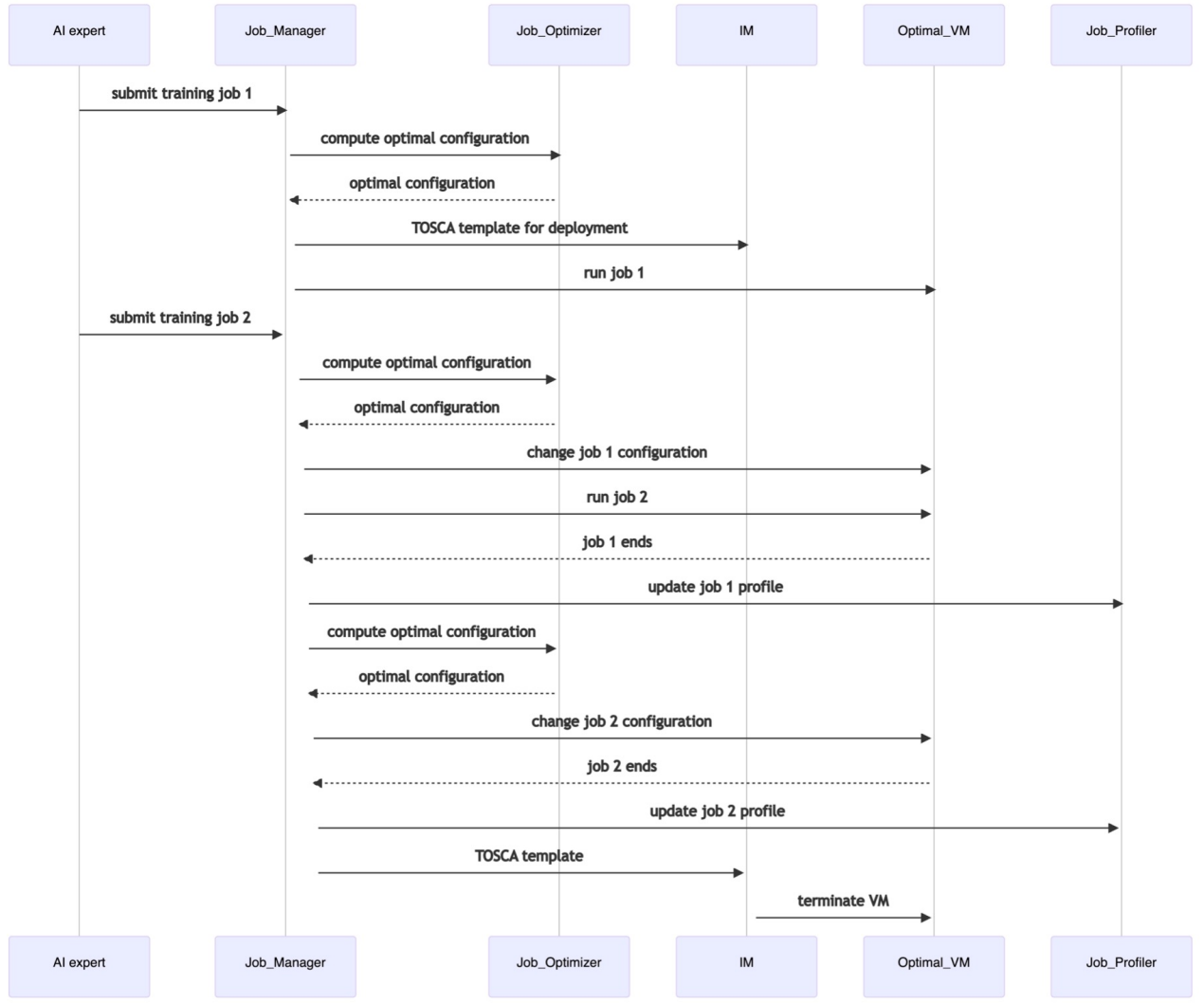
request to the Job Profiler to determine the execution time of a few epochs on a profiling node. Once the job1 performance profile is available, the optimal configuration is identified by the Job Optimizer and job1 execution starts. In the following, the end user submits job2. In this case, job2 was already run in the system and hence the profiling phase can be skipped. The Job Manager directly invokes the Job Optimizer which decides to change the configuration of job1 and to start the execution of job2 on the same node. When job1 ends, job1 performance profile is updated and job2 configuration is changed by the Job Manager (e.g., allocating all the GPUs available on the selected node). When job2 ends, its performance profile is updated.

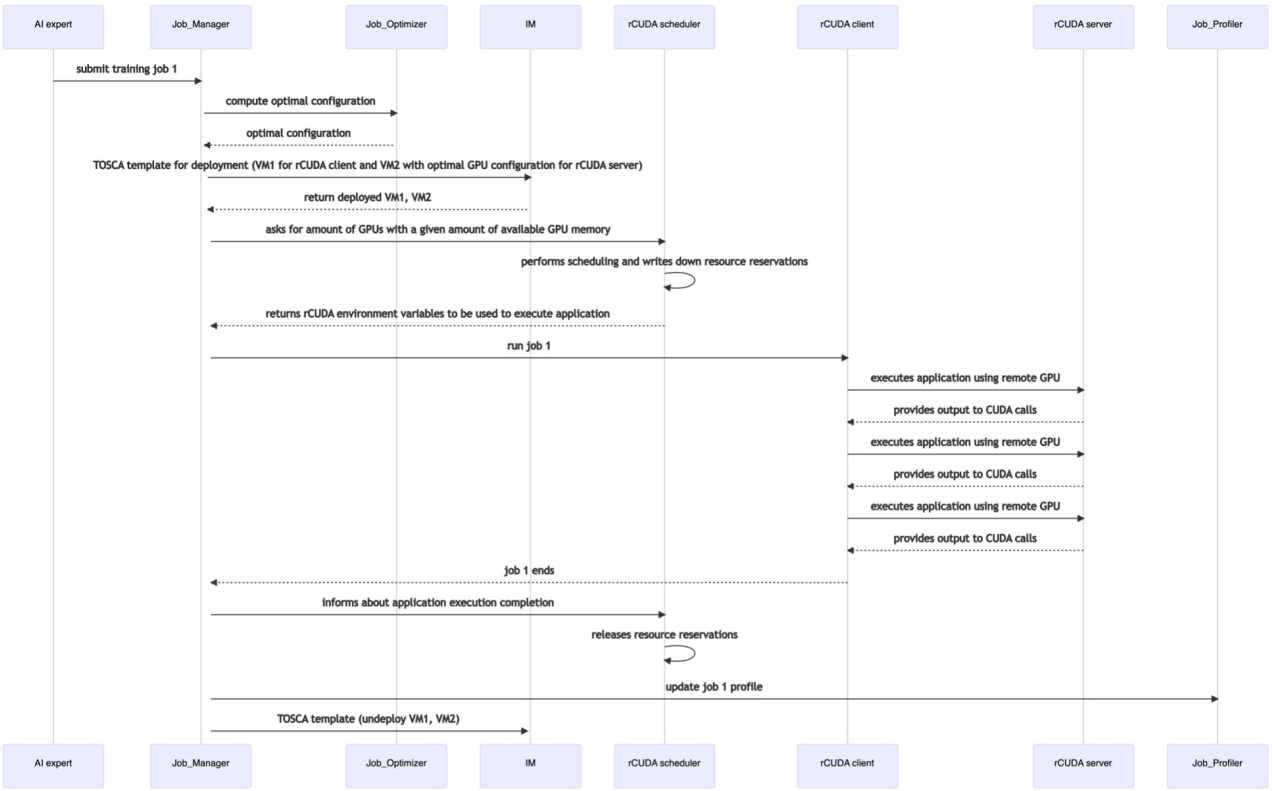
The second sequence diagram illustrates a public cloud deployment scenario under the assumption that the jobs profiles are already available. In that case, the first VM is initially deployed by IM. When also job2 is submitted to the system, the Job Optimizer decides to allocate a different VM which will be shared by job1 and job2. When job1 ends, job2 will use all the available GPUs. After 15 minutes from job2 termination, the VM will be deallocated by IM.

Finally, the last sequence diagram considers a private cloud deployment under the assumption that the job profile is already available but the system is based on a disaggregated resource configuration and GPUs are accessed remotely through rCUDA. In this case, first a VM for the rCUDA client and one with the remote GPU onboard are started by IM. Then, prior to executing the application (let's say job 1), the Job Manager needs to ask the rCUDA scheduler for the appropriate GPUs to be used and the GPU memory demanded by the application. As response to Job Manager request, the rCUDA scheduler provides the right configuration to be used when the application "job 1" is later run with rCUDA. This configuration is provided as the set of rCUDA environment variables to be used during the execution of the application. Finally, once the application ends, the Job Manager contacts the rCUDA scheduler in order to inform it about the completion of job 1, so that the rCUDA scheduler releases the previously allocated resources. As in the previous scenario, if no other jobs are submitted in the next 15 minutes the rCUDA client and server VMs are undeployed.

Sequence diagrams




AI expert
Job_Manager
Job_Optimizer
IM
Optimal_VM
Job_Profiler
submit training job 1
compute optimal configuration
optimal configuration
TOSCA template for deployment
run job 1
submit training job 2
compute optimal configuration
optimal configuration
change job 1 configuration
run job 2
job 1 ends
update job 1 profile
compute optimal configuration
optimal configuration
change job 2 configuration
job 2 ends
update job 2 profile
TOSCA template
terminate VM
AI expert
Job_Manager
Job_Optimizer
IM
Optimal_VM
Job_Profiler



Data flows

The main data items to flow through these scenarios consists of the description of the configuration of the system (physical nodes in case of private clouds, candidate VMs in case of public cloud), the TOSCA recipes of the VMs to be started, the jobs performance profile data and jobs due dates. In particular, physical nodes are characterized by the number of free and total GPUs available. In case of public clouds, running VMs are characterized by similar data while candidate VMs types are associated also by the hourly cost. Jobs profiling data include the time to perform one epoch under a given configuration (including the batch size) and the associated cost (which includes energy and cooling cost of the system) in case of private clouds. Notice that in this scenario it is possible to take advantage of remote GPU virtualization in order to reduce the amount of GPU resources required by concurrently sharing the available GPUs among multiple applications.

A.11 Privacy preserving continuous training

Summary of the use case

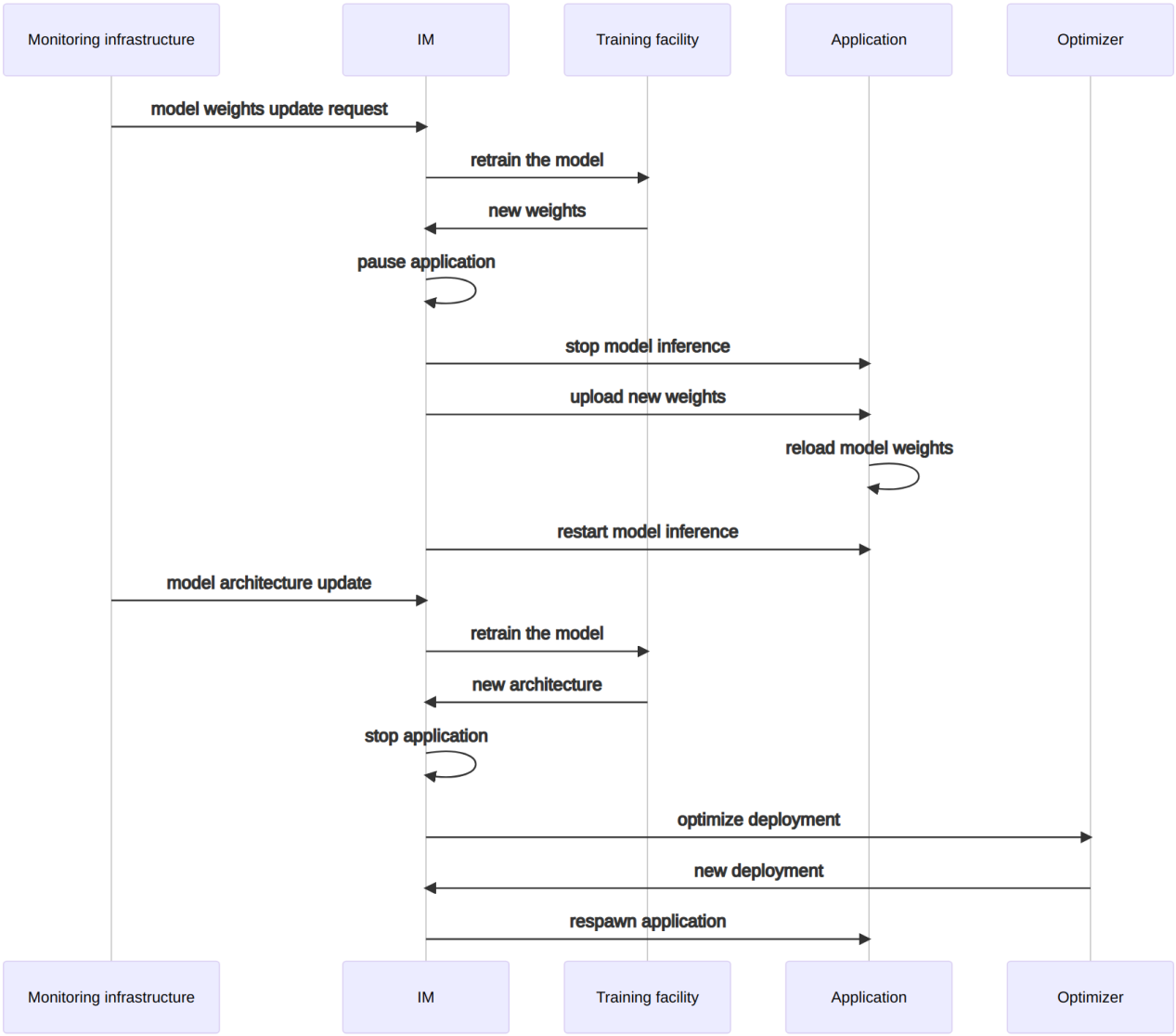
ID	RUN-UC7
Title	Continuous training
Priority	Must have
Actors	Application Manager

Pre-conditions	A new model is available and the monitoring system has triggered an update of the application.
Post-conditions	The application is updated with the new model, either because the weights of the model have been updated or because the new application has been reoptimized and redeployed.

Description of interactions

Based on some monitoring rule the retraining of a model is triggered, the IM triggers the training procedure and once the model is available it updates the application. The model could be trained on demand or it could be already available, this is totally transparent to the IM. Once the new model is available, the IM has two options; if the retraining has just updated the model weights these are reloaded in the application after pausing the inference and triggering a weight reload. If a totally new model is available the overall application deployment is re-optimized and the new application is redeployed by the IM.

Sequence diagrams



Data flows

The Monitoring system triggers the IM for a model update. The IM requests the new model to the Training Facility which returns a new model. If the new model is just a new set of weights, the application is stopped and the weights are reloaded. If the model is a totally new one the application is replaced after optimization.

A.12 Application reconfiguration

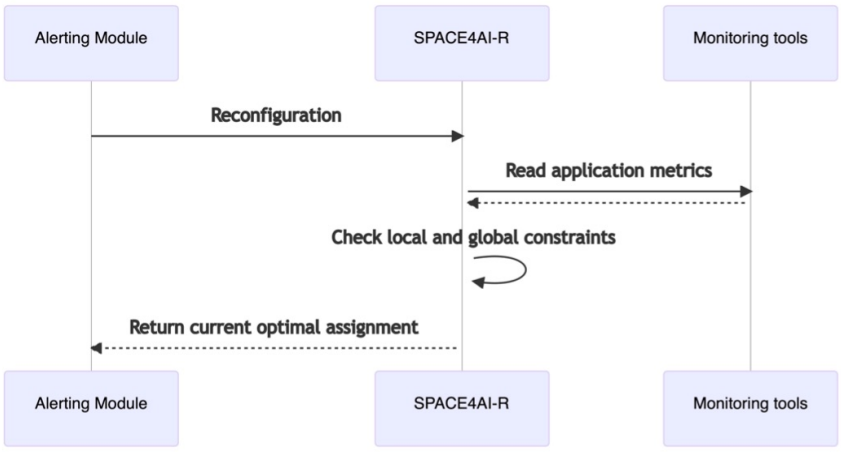
Summary of the use case

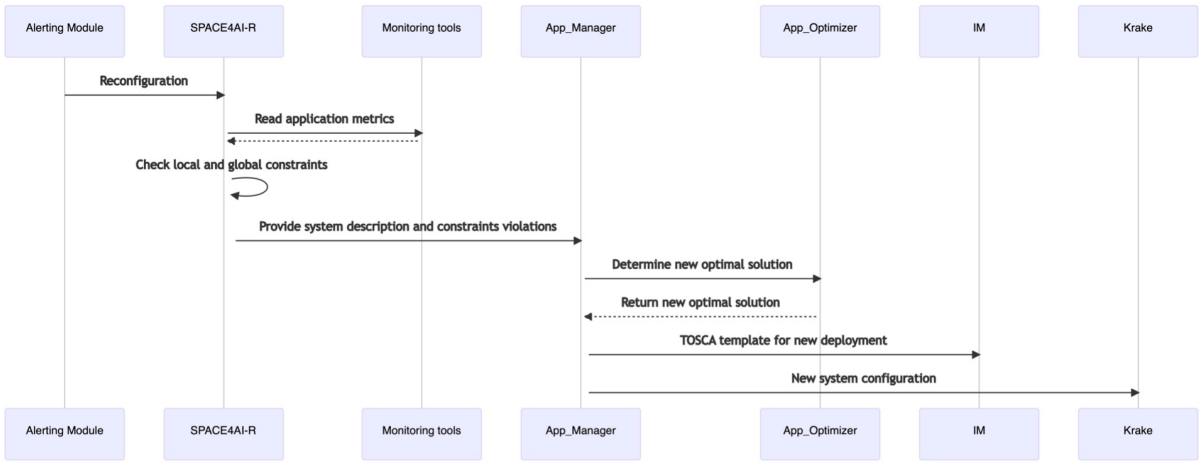
ID	RUN-UC8
Title	Application reconfiguration
Priority	Must have
Actors	Monitoring Infrastructure alerting module
Pre-conditions	AI application components are running with given configurations, on a system involving edge and cloud resources. The system status, including components response times and execution costs, should be periodically reevaluated to account for load variations. These, inducing resource saturation or underutilization, may have a strong impact on costs and/or the fulfillment of QoS requirements.
Post-conditions	SPACE4AI-R determines a new optimal solution for the system under study, adapting the current assignment in terms of application components configurations and resources to the current load. The new optimal solution may include changing the components configuration, and/or scaling the number of cloud VMs used to run specific components and migrating some components from edge to cloud or vice versa.

Description of interactions

The application reconfiguration process is executed periodically to monitor the status of the system and to react to load variations at runtime. This process is in charge of the SPACE4AI-R tool which includes two main modules: the Application Manager and the Application Optimizer. Moreover, SPACE4AI-R relies on IM to deploy additional physical resources on cloud or edge and on Krake for component migrations. *SPACE4AI-R* invokes the monitoring tools, which provide a description of the current system status, in terms of execution time of the running components and resource consumption (e.g., CPU utilizations). According to this information, *SPACE4AI-R* checks whether QoS local and global constraints (namely, requirements on the maximum admissible response times of single components or sequences of components) are satisfied. If any constraint is violated, the application manager invokes the application optimizer to determine a new optimal solution, adapting the current one to the actual load, and providing the updated deployment description to the Infrastructure Manager (IM) server. The IM server deploys the optimal solution on the target system, or invokes Krake to perform the component migrations, if required.

Sequence diagrams





Data flows

The main data flows involve the application metrics, required by the SPACE4AI-R tool and provided by the Monitoring tools, the TOSCA template, which is provided to the IM server to deploy the required configurations on the target system, and Krake application resource manifest file, provided to Krake in order to perform the required components migrations.

A.13 Trusted Execution Environments

Summary of the use case

ID	SEC-UC1
Title	Provide a secure execution environment for AI applications
Priority	High
Actors	Application End User

Pre-conditions	File as well as network encryption is mandatory while running a process in an enclave is optional and based on the hardware capability. Applications/processes need to be sconified, i.e., are either provided as Alpine Linux binaries as docker containers or available as source code for cross compilation. Furthermore, security policies must be provided that define what files and network connections should be encrypted.
Post-conditions	The application code as well as its data read from a file system (data at rest) or network connection are confidentiality as well as integrity protected.

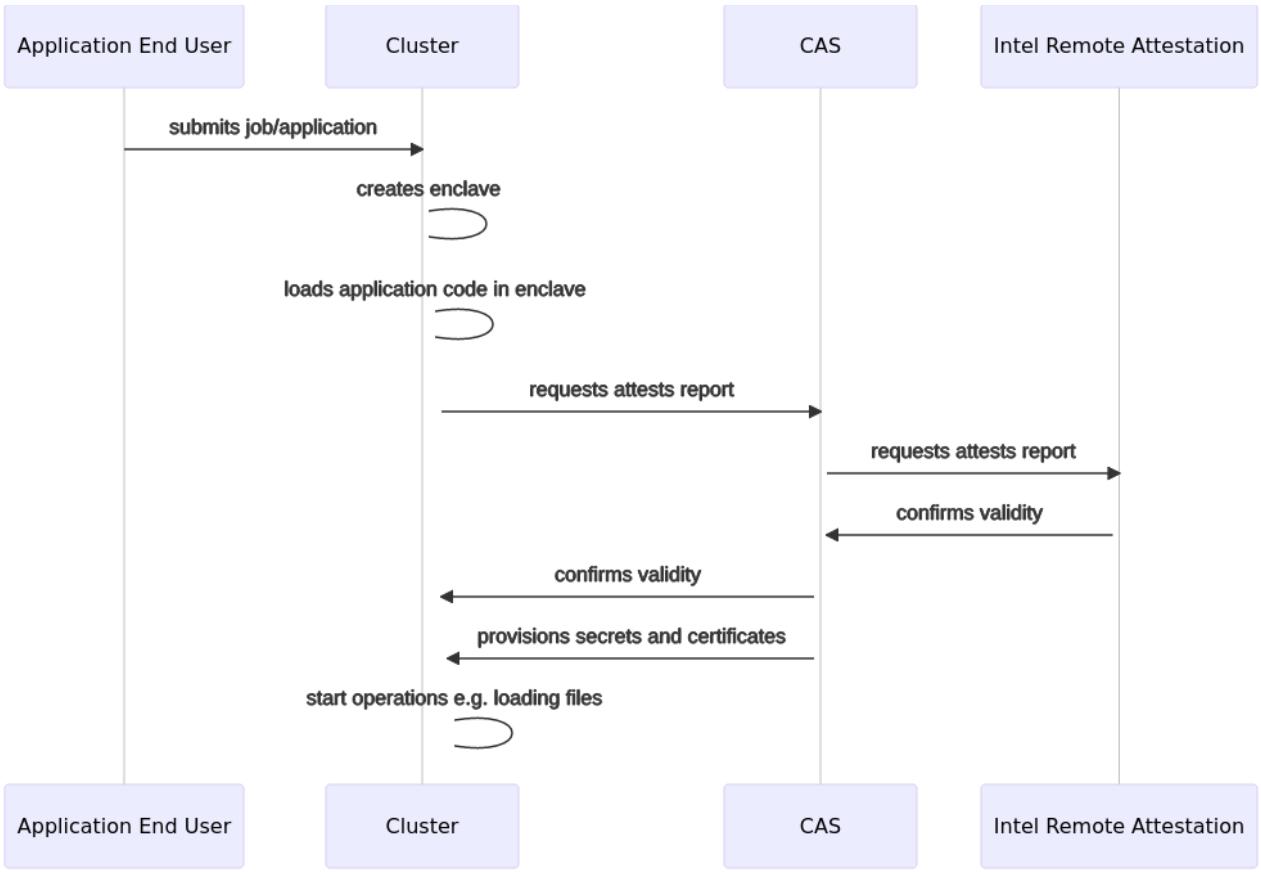
Description of interactions

The application is launched. The launch consists of two steps: First, the binary will launch the starter code which allocates the enclave memory (a trusted compartment). The enclave memory will then be filled with the application code and data that we will refer to as enclave in the following. Once the allocation succeeded, the application code as well as data will be moved into the enclave memory. The application code will then be attested by creating a measurement in order to ensure that the code was not modified while loading into the trusted compartment. Furthermore, using the TEE's instruction of the CPU vendor, the hardware capabilities are checked to ensure that the enclave will run indeed on genuine hardware which provides the necessary protection. Once these tests succeed, the application code within the enclave will be launched, i.e., the enclave is started.

After the enclave start, the application connects to a central entity CAS which is a **C**onfiguration and **A**ttestation **S**ervice for secret and certificate provisioning purposes etc. After verification of the application's identity, the service will provide the application with the necessary secrets in order to perform encryption as well as decryption operations on either file system or network level in order to read/write from/to files and send network packets in a secure manner.

Note that the information about what encrypted volume to encrypt/decrypt and to what other services the applications are allowed to establish connections to is defined through so called security policies stored at CAS. The information is further partly extracted from IM etc. during the deployment of applications onto the infrastructure.

Sequence diagrams



Data flows

The attestation workflow is bidirectional between the enclave and the CAS. Furthermore, secret injection/provisioning is done in a similar manner after successful attestation of the application/process.

After a successful application setup, the communication is mostly between the parties defined in the same policy stored within CAS which includes storing and reading information from encrypted volumes using the file protection shield etc.

A.14 Secure Networks

Summary of the use case

ID	SEC-UC2
Title	Microsegmentation in the 5G Local Breakout
Priority	Must Have if using 5G Local Breakout
Actors	Application Manager, Infrastructure Providers
Pre-conditions	Service is deployed on a 5G Mobile Edge Computing (MEC) platform using Local Breakout (3GPP N6 interface). Client nodes are deployed on 5G devices.
Post-conditions	User devices can connect only to the allowed AI-SPRINT service.

Description of interactions

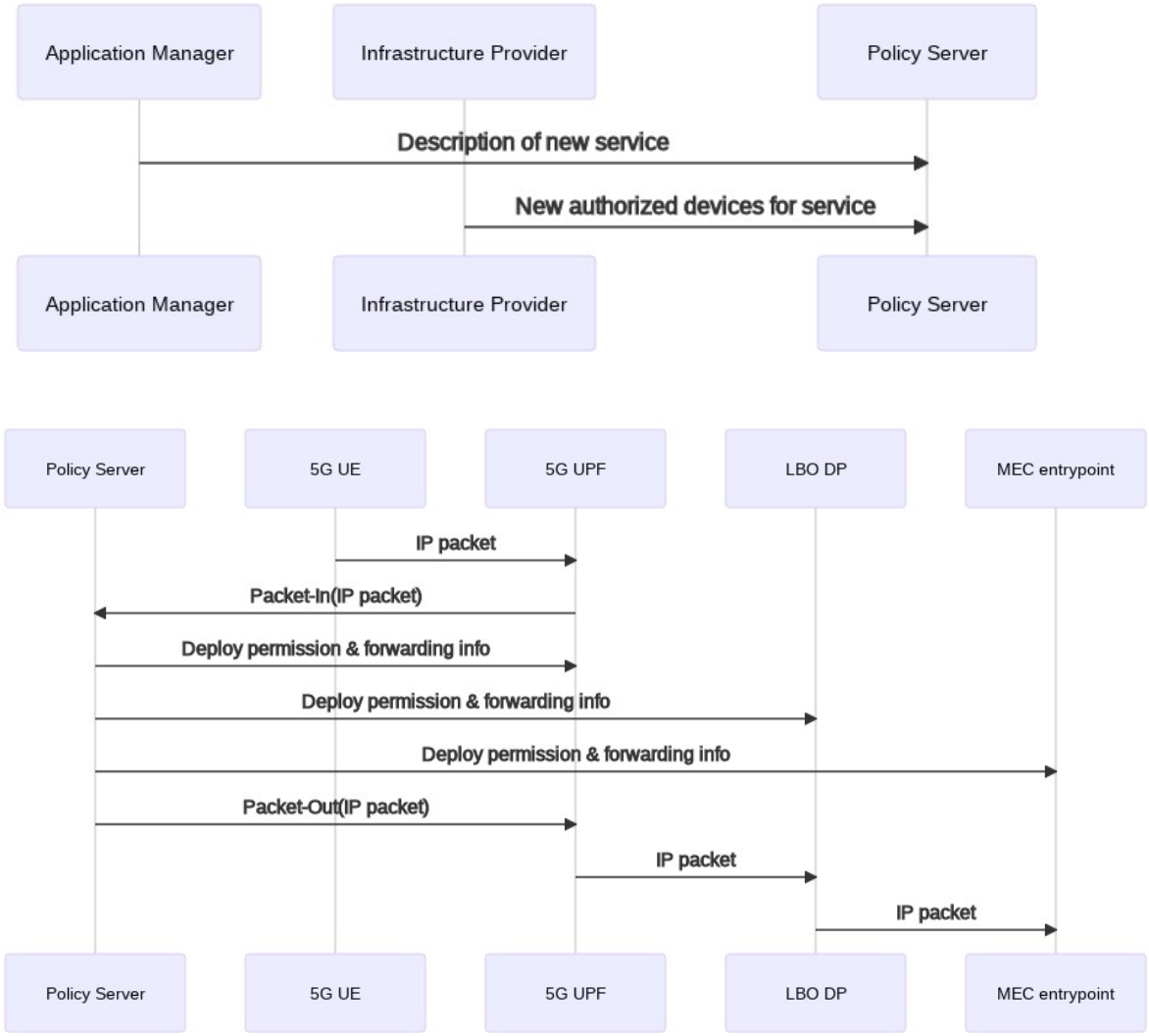
There are two phases. In the first phase, shown in the first sequence diagram, the Application Manager and the Infrastructure Provider provide the descriptions of the new service and of the devices allowed to access it. In the second phase, shown in the second sequence diagram, the service is already deployed and a 5G device tries to access it.

In the first phase, the Application Manager provides an application description file with the indication of the name of the service and the service entry point to the Mobile Edge Computing platform, typically an IP address and a TCP port number. Then, the Infrastructure Provider enrolls each 5G User Equipment (UE) that is allowed to use the service by adding the identifier of the SIM card to the Policy Server.

Later, in the second phase, when the 5G UE first accesses the service by sending an IP packet, typically a TCP SYN packet, to the address of the service entry point, the node executing the 5G User Plane Function (UPF) sends a copy of the packet to the Policy Server. The Policy Server verifies that the packet comes from an IP address assigned to an authorized 5G UE and deploys traffic routing and filtering rules between the UPF and the service entry point. These rules are deployed to any switch or router between the UPF and the MEC (indicated in the Sequence Diagram as Local Breakout Data Plane nodes) and on the nodes executing the UPF and the MEC.

Once the rules are deployed on the nodes, the packet is forwarded to the entry point.

Sequence diagrams



Data flows

In the first phase, data flow is in one direction, from the Application Manager and from the Infrastructure Provider to the Policy Server. Such data consists in the description of the service (e.g., a Kubernetes deployment file) and in a list of authorized devices (e.g., a list of SIM identifiers).

In the second phase, the data flow consists in a 5G UE performing a TCP active open to the service endpoint. This TCP segment exits the 5G network at the UPF located at the MEC premises (this operation is called “Local Breakout”) and is routed to the entry point of the MEC cluster through the 3GPP-defined interface N6. Such an interface can either be an unstructured IP network or can be an IP network transporting point-to-point overlay tunnels between the UPF and the MEC. In the AI-SPRINT network model, we assume that it is an SDN network that can be configured by using an SDN controller and that it consists of a number of Data Plane nodes that can be programmed using the P4 language. When the TCP packet reaches the UPF interface and no rule is already in place to forward it, it is sent to the Policy Server, which also serves as the SDN controller. In case the packet is allowed, the Policy Server deploys the necessary rules to forward it to the service and to forward any answer from the service to the 5G UE. After successful deployment of the traffic filtering rules, the traffic is bidirectional and does not involve the Policy Server.

A.15 Secure Boot

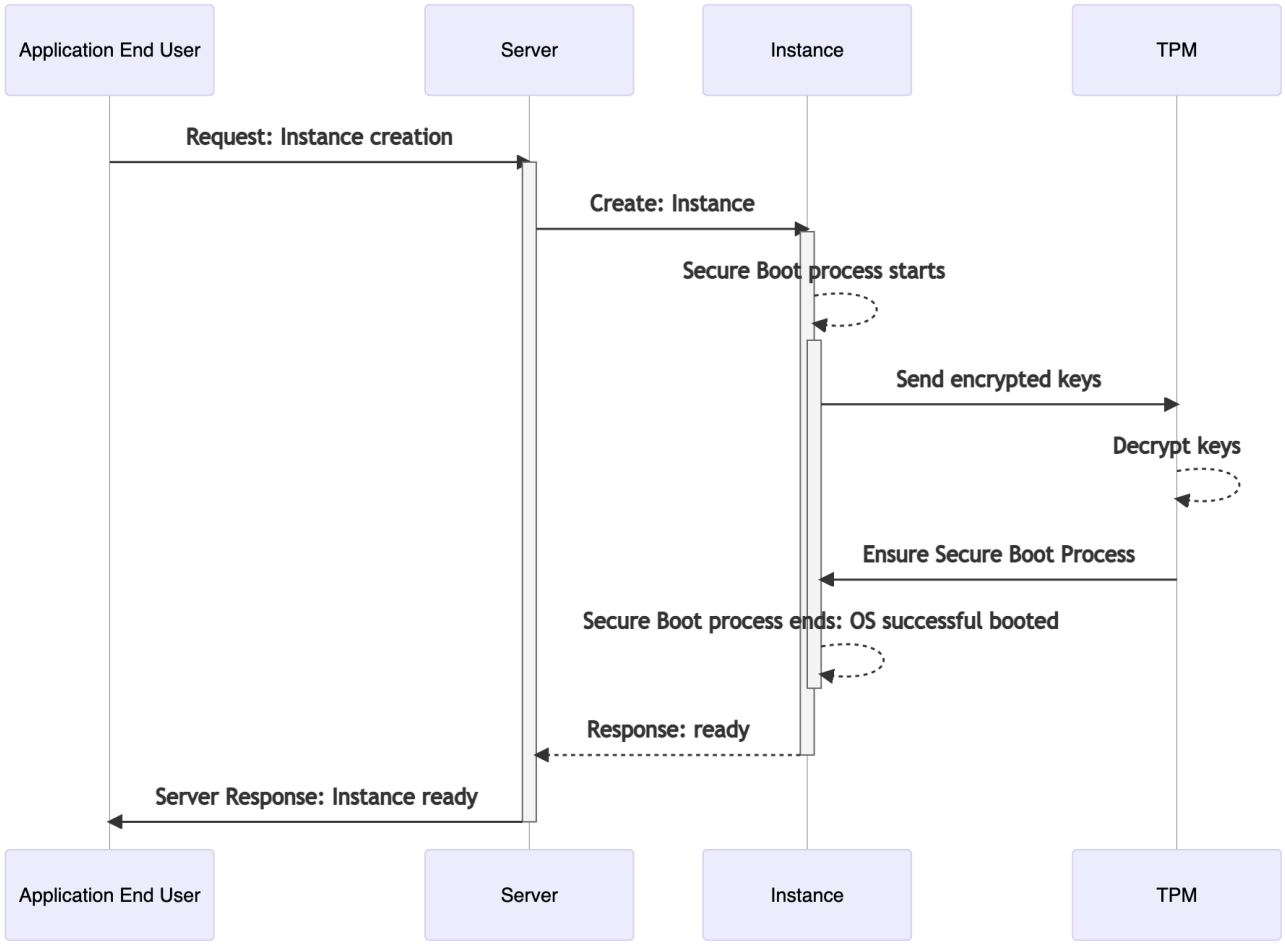
Summary of the use case

ID	SEC-UC3
Title	Ensuring Secure Boot processes and establishing TPM (Trusted Platform Module) usage
Priority	Must have
Actors	Application End User
Pre-conditions	Secure Boot needs UEFI firmware to be enabled, servers need a TPM to use hardware based TPM, without both the system could be more likely infected by malware. A boot process is performed successfully even if the system is infected.
Post-conditions	By setting up Secure Boot, the system is secured by signature verification of every booting software. The TPM, on the other hand, extends Secure Boot with cryptographic services like hashes or encryption. The TPM stores these hashes or encryption keys in a separate memory. The boot process of an infected or incorrectly configured system is interrupted.

Description of interactions

The process of Secure Boot starts each time an instance like a GPU-node or a VM is started. In the process Secure Boot checks every signature against a specific private key, which the developer of the booting software used to sign the code beforehand. The public key of the matching private key will be used to verify the origin of the software. This happens before anything else in the boot process is executed. If a check fails, the entire boot process is aborted. If the checks are successful, the firmware hands over control to the booted operating system. It is important to note that both the Secure Boot process and the TPM must be well configured to ensure adequate security.

Sequence diagrams



Data flows

The data that is sent consists mainly of keys and certificates that are bound locally to the cloud, edge or VM instance side. Since Secure Boot mostly uses asymmetric cryptography, the keys are stored locally in "signature databases" which are integrated in the UEFI firmware. These keys are only transferred between these different databases, the TPM and the booted components. No data from the Secure Boot process is published outside the process or instance.