Sodalite

SOftware Defined AppLication Infrastructures managemenT and Engineering

Intermediate Implementation and Evaluation of the SODALITE Platform and Use Cases

D6.3

ADPT

31.1.2021

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825480.

Deliverable data			
Deliverable	Intermediate implementation and evaluation of the SODALITE platform and use cases		
Authors	Kamil Tokmakov (USTUTT), Ralf Schneider (USTUTT), Dennis Hoppe (USTUTT), Kalman Meth (IBM), Gad Maor (IBM), Elisabetta Di Nitto (POLIMI), Piero Fraternali (POLIMI), Rocio Nahime Torres (POLIMI), Saloni Kyal (POLIMI), Giovanni Quattrocchi (POLIMI), Paul Mundt (ADPT), Yosu Gorroñogoitia (ATOS), Lucas Pelegrin (ATOS), Jorge Fernández (ATOS), Dragan Radolović (XLAB), Mihael Trajbarič (XLAB), Zoe Vasileiou (CERTH), Alfio Lazzaro (HPE), Indika Kumara (UVT/JADS)		
Reviewers	Alfio Lazzaro (HPE) Indika Kumara (UVT/JADS) Daniel Vladušič (XLAB)		
Dissemination level	Public		
History of changes	Name	Change	Date
	Kamil Tokmakov Paul Mundt	Outline created	07.12.2020
	All	Preliminary contributions	08.01.2021
	All	Refinements	08.01.2021
	All	Internal review	20.01.2021
	All	Finalization of contributions	28.01.2021
	Kamil Tokmakov Paul Mundt	Final version	29.01.2021

Acknowledgement

The work described in this document has been conducted within the Research & Innovation action SODALITE (project no. 825480), started in February 2019, and co-funded by the European Commission under the Information and Communication Technologies (ICT) theme of the H2020 framework programme (H2020-ICT-16-2018: Software Technologies)

Table of Contents

List of Figures

List of Tables

Executive Summary

In this deliverable, we report on the implementation progress in the second year of the project, its culmination in the second prototype of the SODALITE platform, and the evaluation of the release through a combination of technical KPI assessment and validation by the project's three demonstrating use cases.

The key contributions and achievements with respect to the SODALITE platform are:

- The introduction of the Edge as a unique infrastructure, as utilized by the Vehicle IoT UC. This has included the extension of the Cloud testbed to connect with a new Edge testbed in order to support Cloud-to-Edge deployments and experimentation. The Edge testbed includes a variety of hardware configurations and heterogeneous accelerators, managed as a self-contained Kubernetes cluster.
- Quality controls for all software components have been implemented through SonarCloud, with each component required to pass a quality gate.
- Development updates for individual SODALITE layers are presented below:
    - Semantic Modelling Layer: advanced features (e.g. design optimisation and Ansible models) were developed and released, the components were deployed and fully integrated. The SODALITE IDE has been improved for modules/extensions in the AADM, and extended to support the Ansible DSL, while the Semantic Reasoner and Semantic Knowledge Base have been extended with user management, support for TOSCA policies, and enriched context-assistance.
    - Infrastructure as Code Management Layer: During this period, some of the components developed during the first year were refactored and significantly improved. A new platform discovery service and application optimizer (MODAK) were released and integrated. IAM and secrets management have also been added.
    - Runtime Layer: Monitoring has been extended to include a visualization dashboard and alert manager, as well as Edge-based exporters. An alerting rules service has been developed to provide dynamic discovery of alerting rules. Metric exporters are dynamically discovered through Consul and Kubernetes.

The key contributions and achievements with respect to the evaluation of the use cases are:

- Snow UC: All use case components were released as scheduled.
- Clinical Trials UC: All use case components were released as scheduled. The original processing pipeline was extended by additional components.
- Vehicle IoT UC: All use case components were released as scheduled. While the Y1 focus has been on developing use case components in Cloud, the Y2 focus has shifted towards Edge-based deployment through a Kubernetes cluster. Edge-based metric exports have been developed and tied into the SODALITE run-time monitor, allowing for direct reconfiguration of use case components at the Edge by the Cloud-based refactorer.

Glossary

Acronym	Explanation
3D	Three Dimensional
AAI	Authentication and Authorization Infrastructure
AADM	Abstract Application Deployment Model
ALPR	Automatic License-Plate Recognition
AOE	Application Ops Expert The equivalent process from the ISO/IEC/IEEE standard 12207 Systems and software engineering — Software life cycle processes is Operation processes and maintenance processes
API	Application Program Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration/Continuous Delivery
CLI	Command-Line Interface
CRI	Container Runtime Interface
CSAR	Cloud Service Archive
CT	Computer Tomography
CV	Computer Vision
DEM	Digital Elevation Model
DICOM	Digital Imaging and Communications in Medicine
DMI	Daily Median Image
DSL	Domain-Specific Language
DXA	Dual Energy X-ray Absorptiometry
EAR	Eye Aspect Ratio
ECG	Electrocardiogram
EMF	Eclipse Modelling Framework
EXIF	Exchangeable Image File Format
FaaS	Function as a Service
FEM	Finite Element Method
FOV	Field of View
FPGA	Field-Programmable Gate Array
GA	Grant Agreement
GCP	Google Cloud Platform
GDPR	General Data Protection Regulation

GPU	Graphics Processing Unit
HPC	High Performance Computing
HPVM	High Performance Virtual Machine
IaC	Infrastructure as Code
IAM	Identity and Access Management
IaaS	Infrastructure-as-a-Service
IDE	Integrated Development Environment
IoT	Internet of Things
IPMI	Intelligent Platform Management Interface
ITK	Insight Segmentation and Registration Toolkit
JSON	JavaScript Object Notation
KB	Knowledge Base
LRE	Lightweight Runtime Environment
M<X>	Month <X> of the project
M2T	Model-to-Text
MCA	Marching Cubes Algorithm
MIGR	Mountain Image Geo-registration
ML	Machine Learning
MPI	Message Passing Interface
MRI	Magnetic Resonance Imaging
MS<X>	Milestone X
MTU	Maximum Transmission Unit
NIC	Network Interface Controller
OCI	Open Container Initiative
OCR	Optical Character Recognition
PERCLOS	Percentage of Eyelid Closure
QoS	Quality of Service
QE	Quality Expert The equivalent process from ISO/IEC/IEEE standard 12207 Systems and software engineering — Software life cycle processes: Infrastructure management and Configuration management processes
RDF	Resource Description Framework
RE	Resource Expert The equivalent process from ISO/IEC/IEEE standard 12207 Systems and software engineering — Software life

	cycle processes is Quality Management and Quality assurance processes
REST	Representational State Transfer
SLA	Service Level Agreement
SVC	Support Vector Classifier
SVM	Support Vector Machine
ToR	Top-of-Rack
TOSCA	Topology and Orchestration Specification for Cloud Applications
TPU	Tensor Processing Unit
UC	Use case
UDJ	Universal Data Junction
UGI	User Generated Images
UML	Unified Modeling Language
VIN	Vehicle Identification Number
VM	Virtual Machine
VPN	Virtual Private Network
VTK	Visualization Toolkit
WP<X>	Work Package X
Y<X>	Year <X> of the project
YAML	YAML Ain't Markup Language

1 Introduction

The objectives of work package WP6 are integration, evaluation and validation of the SODALITE framework as detailed and specified in WP2. The components to be integrated are developed across work packages WP3, WP4, and WP5. A combination of multi-tiered external and internal evaluation is done using the SODALITE use cases. This deliverable reports therefore on the current status of the SODALITE platform and its use cases to assess the overall progress made up to the second year of the project.

The previous deliverable, D6.2[1], which was due in project month M12, reported the status of the initial implementation of the SODALITE components and their integration into the First SODALITE Prototype, the initial implementation and the status of the demonstrating use cases. It also provided detailed information about the advancements made with respect to the SODALITE development environment, which includes the HPC and Cloud testbeds, the SODALITE repository and the Continuous Integration/Continuous Delivery (CI/CD) pipeline for automated components integration and testing.

This document is an update of D6.2 and reports on the intermediate implementation of the use cases, SODALITE components, their integration into the SODALITE platform (MS6 - Second Prototype), as well as their evaluation and validation. More specifically, D6.3 provides updates on the development environment, with the most prominent being an introduction of Edge testbed, enhanced coverage of target platforms, improved platform integration with CI/CD and integration of tools for software quality measurement. Furthermore, this report updates the development and integration status of SODALITE components (new components were introduced in Y2) as well as demonstrating use cases. Lastly, compared to the previous deliverable, the platform validation and evaluation are significantly more extensive. Apart from platform evaluation performed by the use case owners, it provides evaluation details additionally done by the external users, and reports on KPIs achieved by M24.

This deliverable has been developed in parallel and coherently to WP2, WP4 and WP5 deliverables D2.2, D4.2, D5.2, another WP6 deliverable D6.6 and to the work developed in WP3 as part of the second project year.

Throughout the document, we are using the terms Application Ops Experts (AOE), Resource Experts (RE) and Quality Experts (QE). The following table provides a mapping between these roles and the processes defined in the ISO/IEC/IEEE standard 12207 Systems and software engineering — Software life cycle processes:

SODALITE Roles	ISO/IEC/IEEE standard 12207 processes
Application Ops Experts (AOE)	Operation processes and maintenance processes
Resource Experts (RE)	Infrastructure management and Configuration management processes
Quality Experts (QE)	Quality Management and Quality assurance processes

1.1 Structure of the Document

The structure of this deliverable is as follows:

- The remaining part of Section 1 highlights updated SODALITE architecture and its components, as well as presents the objectives and status of the MS6 - Second Prototype.
- Section 2 provides a description and updates of the development environment, which includes the HPC and Cloud testbeds, the repository and the CI/CD pipeline. Additionally, it

highlights the software quality of the developed components and tested platforms outside the testbeds.

- Section 3 outlines the development and integration status of the MS6 - Second Prototype. The components of the Prototype are described in detail in the deliverable D6.6 - SODALITE framework - Second version[2].
- Section 4 provides the development status of the SODALITE demonstrating use cases.
- Section 5 highlights the validation and evaluation of the MS6 - Second Prototype by describing how demonstrating use cases and external users have been using the features offered by the SODALITE platform.

1.2 SODALITE Architecture

For greater clarity, we reproduce a synopsis of the SODALITE architecture that is described in Deliverable D2.2[3]. For the details of the functional description, inputs, outputs and dependencies of each component, please refer to the architecture section (Section 4) in D2.2.

SODALITE aims to provide developers and infrastructure operators with tools that abstract their application and infrastructure requirements to enable simpler and faster development, deployment, operation and execution of heterogeneous applications on heterogeneous, software-defined, high-performance and cloud infrastructures. To this end, SODALITE aims to produce:

- a pattern-based abstraction library that includes application, infrastructure, and performance abstractions;
- a design and programming model for both full-stack applications and infrastructures based on the abstraction library;
- a deployment framework that enables the static optimization of abstracted applications onto specific infrastructure;
- an automated run-time optimization and management of applications.

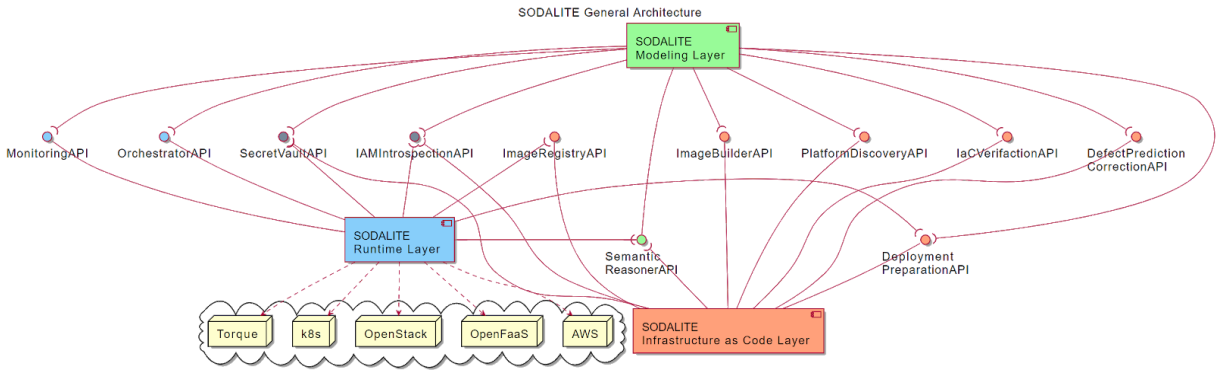

Figure 1 - SODALITE overall Architecture

The SODALITE platform is divided into three main layers, each covered by a separate work package (WP). These layers are the Semantic Modelling layer (WP3), the Infrastructure as Code Management layer (WP4), and the Runtime layer (WP5). Figure 1 shows these layers together with their relationships.

1.2.1 SODALITE Semantic Modelling Layer

Figure 2 shows the internal architecture of the SODALITE Modelling Layer.

IDE is a graphical environment providing full support for the authoring of resource and deployment models at design time and the management of deployed applications at runtime. In particular, the IDE provides a user interface with several DSL editors for the specification of infrastructure resources, the design of the application deployment topologies, component optimization and Ansible models for operation implementations. Users are assisted during the modeling process through context-aware content assistance by getting recommendations and having the model semantically validated. This Smart intellisense of IDE mainly derives from the Semantic Reasoner Knowledge. The IDE also enables users to browse and manage their models stored in the Semantic Knowledge Base.

A set of SODALITE domain ontologies, resulting from the abstract modelling of the related domains (applications, infrastructure, performance optimisation and deployment), will be hosted in a SPARQL-served RDF Triplestore (GraphDB), constituting SODALITE's Semantic Knowledge Base. A dedicated middleware (Semantic Reasoner) enables the exploitation of this repository, mediating for the population of data and the application of rule-based Semantic Reasoning. The interfaces offered by other components, other than Semantic Reasoner, are highlighted. The IDE communicates with other system APIs for the verification and deployment of the abstract model, and the monitoring of the deployment lifecycle.

A more detailed description of the components of this Layer and their development plan is described in Appendix A of the D6.6 (SODALITE Framework - Second Version).

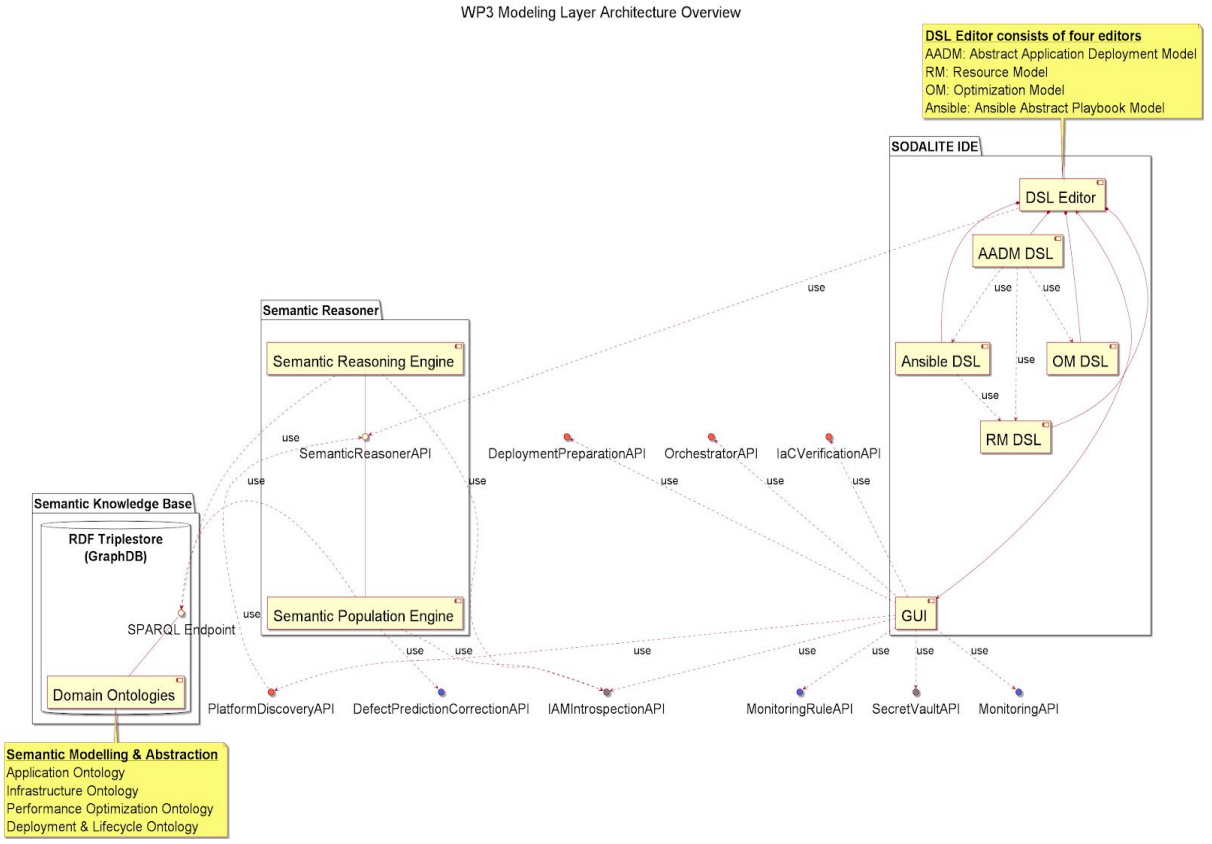

Figure 2 - SODALITE semantic modelling layer components (WP3)

1.2.2 SODALITE Infrastructure as Code Management layer

The components of the SODALITE Infrastructure as Code (IaC) Management Layer are depicted in Figure 3. The Infrastructure as Code Layer (IaC Layer) is the layer that connects the SODALITE modelling layer functionalities to Runtime blueprint execution of the models in the SODALITE Runtime Layer. It offers APIs and data to support the optimization, verification and validation process of both Resource Models (RM) and Abstract Application Deployment Models (AADM).

However, one of the most important tasks of the IaC Layer is preparing a valid and deployable TOSCA blueprint.

In the second year of the project, some of the components were initially released and several were refactored and significantly improved. During this period Platform Discovery Service has been added to the layer's architecture, to expose a REST API which helps to automate the tasks of the Resource Expert by creating a valid TOSCA platform Resource Model (RM) to be stored into the SODALITE's Knowledge Base. These RMs can then be used during the design of the application deployment models (AADM).

In this period Application Optimizer (MODAK) component, exposing a REST API, was released and integrated into the pipeline enabling the SODALITE users to statically optimize the application for a given target execution platform.

Automation of application optimisation on both HPC and cloud systems requiring models used for performance prediction have been improved. SODALITE prepares and uses these models for both pre-deployment (static) performance optimization and runtime (dynamic) performance optimization.

Additionally IAM (Identity and Access Management) API and Secret Vault API have been added and partially integrated into IaC Layer and used by the components that have to protect secrets stored by the user such as Platform Discovery Service and IaC Blueprint Builder.

During development in the second year of the project, a part of the architecture was redesigned which was also reported in the deliverable D4.2[4].

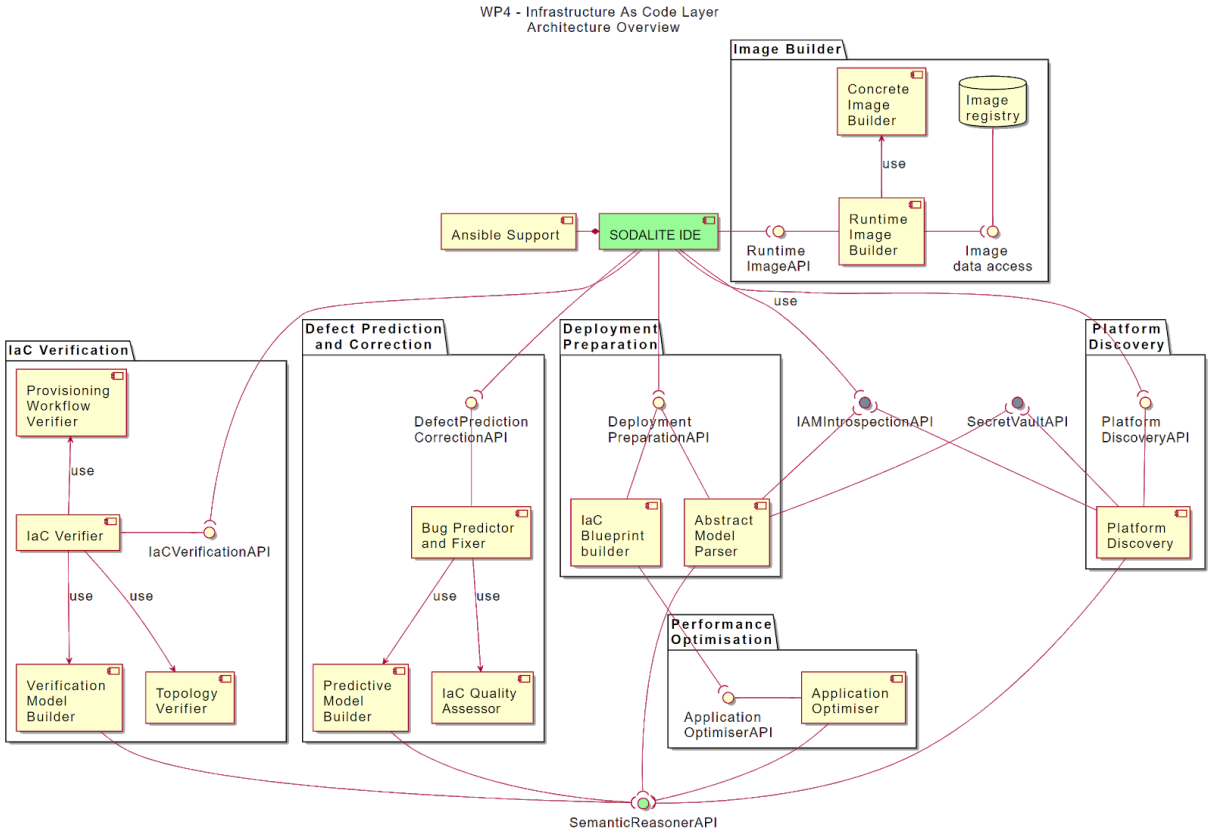

Figure 3 - SODALITE infrastructure as code management layer components (WP4)

1.2.3 SODALITE Runtime layer

The Runtime layer of SODALITE (see Figure 4) is in charge of the deployment of SODALITE applications into heterogeneous infrastructures, its monitoring and the refactoring of the

deployment in response to violations in the application goals. It is composed of the following main blocks:

- Orchestrator. It receives the application to be deployed or re-deployed as a blueprint expressed in TOSCA, deploying the application components on the appropriate infrastructure.
- Monitoring. It monitors the application components and the infrastructure where they are deployed to be used by Refactoring and the interested SODALITE actors.
- Refactoring. It is able to propose a new application model to fulfil the application goals. When it needs to modify the deployment model, it calls the Deployment Preparation, which will trigger the generation of a new blueprint that arrives to the Orchestrator to initiate the redeployment.

The main changes introduced in the runtime architecture w.r.t. the version reported in D2.1 are the following:

- Orchestration block: it is made explicit the drivers supported and integrated within the orchestrator, namely the OpenStack, Torque, Kubernetes, OpenFaaS and the AWS drivers, which interfaces target VMs, HPC schedulers, Kubernetes, OpenFaaS and AWS infrastructures, respectively. It also incorporates additional required interfaces for AAI (e.g. IAMIntrospectionAPI) and for the retrieval of deployment secrets (e.g. SecretVaultAPI).
- Monitoring block: this block includes new components not previously included in the former architecture:
    - Monitoring Dashboard: this frontend provides monitoring specialized visual reports for selected target application components and execution environments. Dashboard uses the MonitoringAPI REST interfaces to query monitoring data.
    - Monitoring Alert Manager: manages defined alerting rules to trigger notifications to subscribers when the rule condition holds for the target monitoring statistics. The AlertingAPI REST interface is used by the Orchestrator, Node Manager and Deployment Refactorer components to subscribe themselves to concrete alerts.
- Refactoring block: the internal interactions among these block components have been made explicit in the architecture, through the REST API exposed by each component.

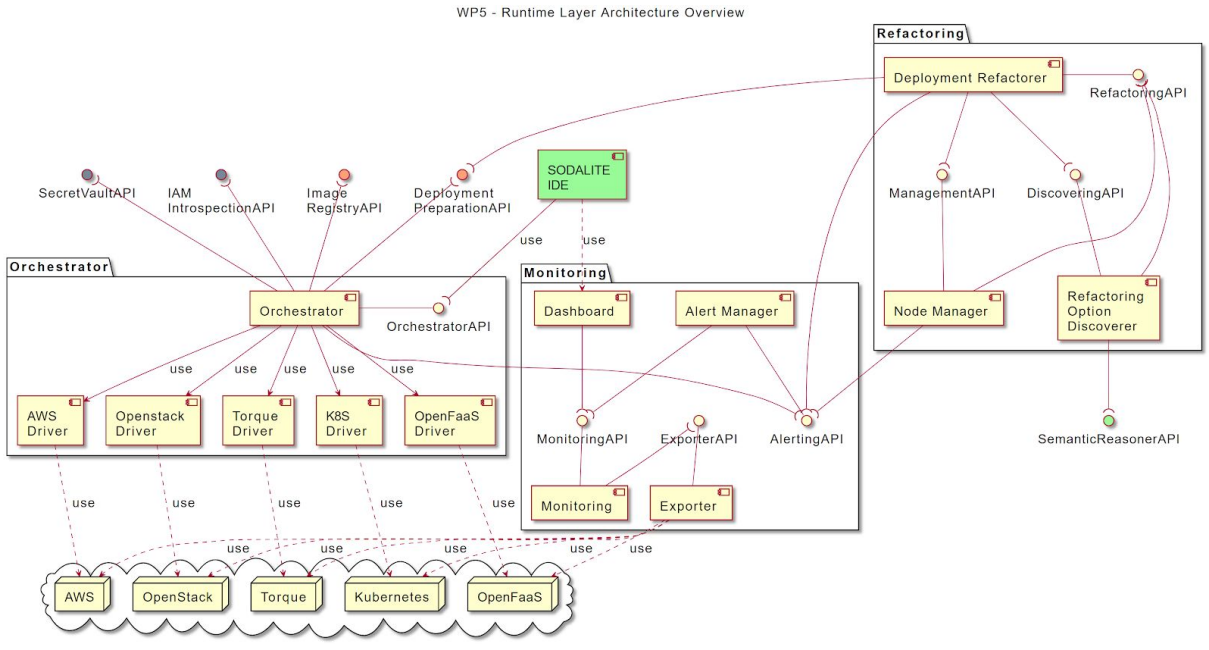

Figure 4 - SODALITE runtime layer components (WP5)

1.3 Objective of the MS6 - Second Prototype

In the MS6 - Second Prototype, all of the SODALITE components are expected to be released as stable versions, and form a more integrated, intermediate implementation of the SODALITE platform that provides the first advanced features. The MS6 - Second Prototype is used to deploy, execute and clearly improve the demonstrating use cases and aims to achieve goals that can be consolidated from the objectives of the SODALITE Architecture layers:

- Semantic Modelling Layer: the intermediate implementation of the semantic models, the repository and the IDE for supporting users in modelling the application and infrastructure.
- Infrastructure as Code (IaC) Management layer: the intermediate version of the deployment preparation for the selected infrastructure management systems and performance optimization; intermediate implementation of the analytics for the quality of the IaC - verification and bug prediction of deployment models.
- Runtime layer: the intermediate implementation of the cross-platform orchestrating tools, collection of monitoring metrics and intermediate version of predictive deployment refactoring.

1.4 Status of the MS6 - Second Prototype

The status of the SODALITE MS6 - Second Prototype at the end of project month M24 is presented in Table 1 and can be summarized as follows:

- The SODALITE development environment has been improved: the Cloud and HPC testbeds are stable and extended with additional functionalities, such as security and larger capacity. The Edge testbed was introduced. CI/CD pipelines are utilised to automatically build, test and deploy most of the components, the source code of which is open and published in SODALITE GitHub repository. Additionally, tools for software quality measurements were integrated.
- In all three main layers of the SODALITE platform (Semantic Modelling, Infrastructure as Code Management, Runtime), the intermediate versions of the components with advanced features have been released and described in deliverable D6.6. Almost all of the components are deployed in the testbeds (Node Manager has been deployed in Azure public cloud; yet to be deployed in the testbed after integration with SODALITE monitoring) and either fully integrated or integrated partially with the whole platform. Identity and Access Management (IAM) components were introduced and partially integrated with the whole platform.
- The three demonstrating use cases of SODALITE have released their components according to the schedule and have evaluated the Second Prototype, which demonstrated the improvements of the use cases, such as reduced effort for deployment code creation, performance increase due to static and runtime optimisation.
- External users (TOSCA experts and experts) have also evaluated the platform in terms of perceived ease of use, perceived usefulness and intention to use. In general, the users preferred SODALITE IDE over other editors for defining deployment models.

Table 1 - Overall status of the development environment, MS6 - Second Prototype and demonstrating use cases at M24

Components	**Status**
Development Environment	**HPC, Cloud and Edge testbeds** were set up and improved.
	Coverage of target execution platforms was extended with support of Kubernetes, AWS, Federated OpenStack,

	Slurm (along with OpenStack, Torque, already offered at M12).
	SODALITE repositories were structured and host the source code for SODALITE components.
	CI/CD server and pipeline were set up to remotely build software artifacts, perform tests, deploy on the testbed and integrate into the platform.
	Software Quality Measurement Tools were introduced
Second Prototype Components	**Semantic Modelling Layer:** advanced features (e.g. design optimisation and Ansible models) were developed and released, the components were deployed and fully integrated.
	Infrastructure as Code Management Layer: the intermediate versions with advanced features (e.g. IaC building with optimisation artifacts, bug prediction and fixing) were developed and released. New components were introduced (MODAK optimiser, Platform Discovery Service). All components are deployed in the testbeds and most of the components are integrated within the layer. Integration into the platform is partial: some components (e.g. IaC Verification components and Platform Discovery - initially released at M24) have not yet been integrated with Modelling Layer.
	Runtime Layer: the intermediate versions with advanced features (e.g. reconfiguration, refactoring and dynamic monitoring) were developed and released. All components are deployed in the testbeds and most of the components are integrated within the layer. Integration into the platform is partial: some components (e.g. Monitoring and Refactoring) have not yet been integrated into IaC and Modelling layers.
	Identity and Access Management (IAM) components were introduced and partially integrated with the whole platform: integration with IaC and Runtime layers have not yet been completed.
Demonstrating Use Cases	**POLIMI Snow:** the components were released as scheduled and presented in Section 4.1.
	USTUTT Virtual Clinical Trial: the components were released as scheduled and presented in Section 4.2. The original processing pipeline was extended by additional components. Due to that, pipeline integration was delayed.
	ADPT Vehicle IoT: the components were released as

	scheduled and presented in Section 4.3. Due to Kubernetes support not being ready in some of the other SODALITE components before M24, parts of the continuous benchmarking and integration have been delayed until Y3.
Platform Evaluation	**POLIMI Snow:** For the Snow Pipeline, we have developed all components and we have deployed them with SODALITE covering 12 from the 17 proposed UML use cases of the platform. We started by modeling the pipeline using the knowledge base-empowered SODALITE IDE passing for the intermediate steps to obtain the TOSCA blueprint verifying that these could actually automate the deployment and configuration of the whole pipeline. Environment intelligence requires sophisticated data acquisition and analysis pipelines, which must scale to large volumes of data, deliver predictions and alerts with severe time constraints. In particular, we tested the NodeManager on the Skyline Extraction component, and it was able to never violate the set SLA for Skyline Extraction during the run experiments, on the contrary the compared rule-based approach obtained 150 violations. In terms of resource consumption, NodeManager exploited 2515 core*second, 19% fewer than the rule-based approach and the average response time of Skyline Extraction was equal to 0.217ms, 40% faster than the rule-based approach. Also, in order to optimize the training of the Skyline Extraction component, tests were executed to benchmark the training process, by comparing the standard TensorFlow container provided on DockerHub and the optimized one provided by MODAK, for now test were executed on ResNet50 architecture, in coming months we benchmark in our Skyline Extraction component. We also tested the reconfiguration by means of the deployment refactorer. For the use case we report 25% of refactoring scenarios are supported, which we plan to improve in coming months by testing different deployment variants. In our tests, thanks to the reconfiguration capability resource usage violations were prevented (e.g., if CPU usage reaches 80%, we move the application to a machine with higher CPU), the extent of success will be measured in further experiments.
	USTUTT Virtual Clinical Trial: the workflow of Clinical UC was executed with the SODALITE platform (MS6 - Second Prototype). Modelling, optimisation and orchestration aspects of the platform were validated and evaluated in the context of the Clinical UC, showing the use case improvements in the following: (1) Originally, Clinical UC workflow was HPC-driven and executed in a single HPC cluster. With SODALITE, the

	workflow execution is distributed across multiple infrastructure targets. (2) The execution of the Solver (Code_Aster) component of Clinical UC in a container was optimised (3% gain in a single-thread was achieved). Furthermore, parallel build and execution became possible, promising further reduced execution time. (3) The workflow was modelled in SODALITE IDE, which reduced the effort for development of the deployment code, compared to Y1 development in TOSCA, as it became easier to integrate new components and avoid possible deployment errors. Additionally, optimised container runtime can be automatically provided based on the optimisation options specified in SODALITE IDE. The uptake of the SODALITE UML UC was increased, covering 10 out of 17 UCs, compared to 3 out of 16 in Y1.
	ADPT Vehicle IoT: During this period, the SODALITE benefits for the vehicle IoT UC have been primarily qualitative. A number of Edge-based metric exporters were developed and integrated with the SODALITE run-time monitor, allowing for the refactorer to actively redeploy and reconfigure deployed applications at the Edge in response to predefined alerting criteria for specific metrics. Applications that through their heterogeneous accelerator use would have failed silently when accelerators were brought outside of their safe thermal operating ranges (a periodic occurrence in passively cooled Edge Gateway enclosures under load) are now detected and reconfigured by the SODALITE refactorer, reducing the occurrence of inference failure in Edge-based AI inference models. During this period, the image builder has also been extended to generate image variants, allowing application container image variants to be generated for different accelerator runtimes, this work has been key in ensuring that optimized versions of applications can be deployed in Y3 in order to move onto benchmarking.
	External users: We evaluated the usability of the SODALITE via a set of controlled experiments with three groups of external users: normal users (students), TOSCA experts, and SODALITE use case owners. We evaluated KPI 3.1 and KPI 3.2, which focus on the usability and effectiveness of the SODALITE platform in creating and modifying deployment models/codes. Each user group had different experiments. The total number of participants was 18 (9 students, 5 TOSCA experts, and 4 use case owners). Overall, the users considered SODALITE very useful and easy to use compared with the other tools.

It should be noted that this deliverable is the third iteration of four deliverables in total within Work Package 6 that report on the status of the SODALITE platform and the integration and evaluation of its use cases at regular intervals between project month M6 and M36. At the end of the last reporting period (at M36), D6.4 will be released - the final deliverable of WP6 and a logical successor to D6.2 and D6.3. There, the information presented will be updated in order to provide the full picture of the use case implementations, all implemented components and their integration into the SODALITE platform, along with their thorough final evaluation.

2 Development Environment

During Y2 period, there were major changes to the development environment in order to 1) facilitate development of advanced features of SODALITE platform and experimentation with various execution platforms on the testbeds, 2) organise the development and integration of SODALITE components into the whole platform automated via CI/CD and 3) improve quality of the components. As such, an extension of testbeds was performed and a larger set of supported infrastructure targets was provided, CI/CD pipeline was established producing deployable artifacts and software quality measurements were introduced.

2.1 Edge, Cloud and HPC Testbeds

Figure 5 presents an updated overview on the testbeds, their components, resources and the supported platforms for the experimentation with cross-system orchestration and monitoring. During Y2 period, Edge testbed was introduced to allow the experimentation with edge resources managed by Kubernetes.

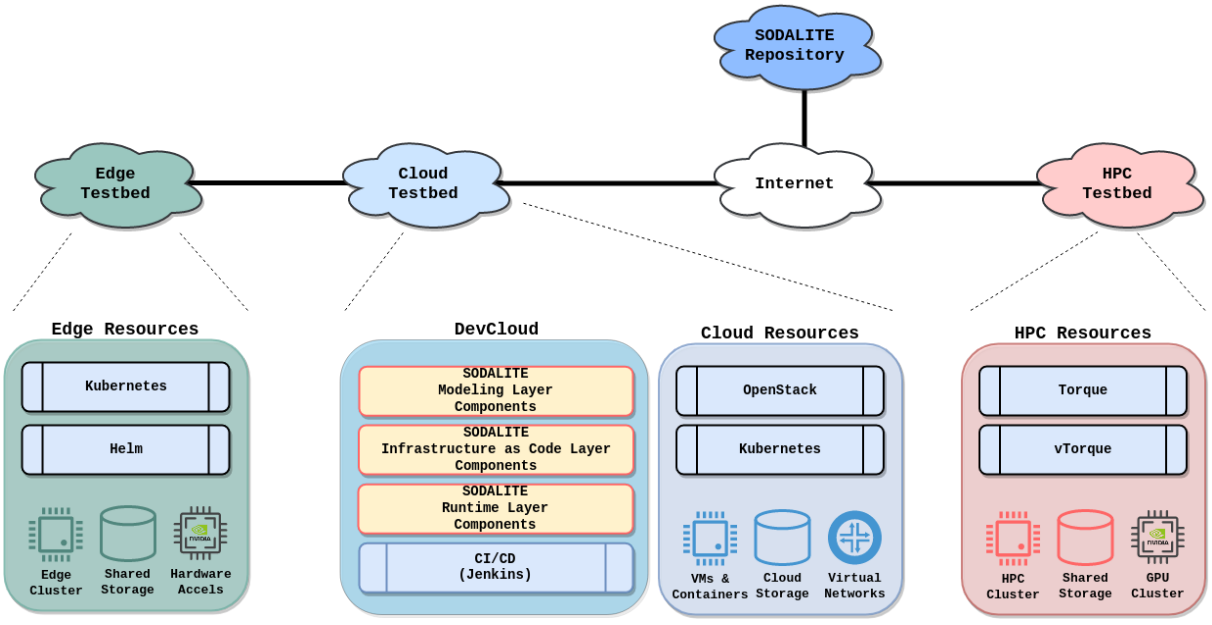

Figure 5 - Cloud, HPC and Edge Testbeds

Edge testbed. The main characteristics of Edge resources are their dynamicity and heterogeneity: the resources can appear and disappear at certain point of time and can vary from common CPU architectures, such as x86 and ARM, to specialised acceleration hardware, such as GPUs, TPUs and FPGAs. Each resource type requires its own unique container runtime that needs to be dynamically provisioned in case new resources are added. Moreover, different accelerators have different performance modes and thermal operating ranges that need to be monitored: stepping outside of these ranges can lead to failures or undefined behaviour. Therefore, edge infrastructures require special resource management and reconfigurations. SODALITE components developed specifically for Edge resource management are trying to address these challenges and their description can be found in the deliverable D5.2[5], Section 4.5.

Edge testbed is provided by ADPT to execute Vehicle IoT UC, and it consists of 4 nodes managed by Kubernetes. These nodes have the following specifications (Table 2):

Table 2 - Specifications of compute nodes in the Edge testbed (ADPT)

Device	**Raspberry Pi 4**	**NVIDIA Jetson Nano**	**Google Coral AI Dev Board**	**NVIDIA Jetson Xavier NX**
CPU	1.5 GHz 64-bit Quad-Core ARM Cortex-A72	1.4 GHz 64-bit Quad-Core ARM Cortex-A57 MPCore	1.5 GHz 64-bit Quad-core ARM Cortex-A53	1.9 GHz 64-bit 6-core NVIDIA Carmel ARM v8.2
GPU	Broadcom VideoCore VI	128-Core NVIDIA Maxwell	Vivante GC7000Lite	384-Core NVIDIA Volta (+ 48 tensor cores)
Other Accelerators	Intel Neural Compute Stick 2 / EdgeTPU (USB)		EdgeTPU	NVDLA x2 Vision Processor x1
RAM	4GB LPDDR4	4GB LPDDR4	1GB LPDDR4	8GB LPDDR4
Performance	24 GFLOPS	472 GFLOPS	4 TOPS	21 TOPS
Storage	SD Card			SD Card + NVMe SSD

Cloud testbed. The Cloud testbed provisions virtualized resources (e.g. virtual machines and containers) managed by OpenStack and Kubernetes. Furthermore, the Cloud testbed hosts the development environment (DevCloud), which contains CI/CD server and deployed SODALITE components. During the reporting period, the stability and security of the testbed were improved.

A VPN was enabled to protect the infrastructure and all elements deployed inside the development environment. Given the decentralized elements that make up SODALITE platform, it is mandatory to ensure the security and privacy of those elements and the communication between them. To achieve that goal two main topics are featured. Firstly to block any incoming public request to the environment and secondly to open exceptions for the ingress layer. An overview on how the ingress layer is installed in the Cloud testbed is presented in Figure 6:

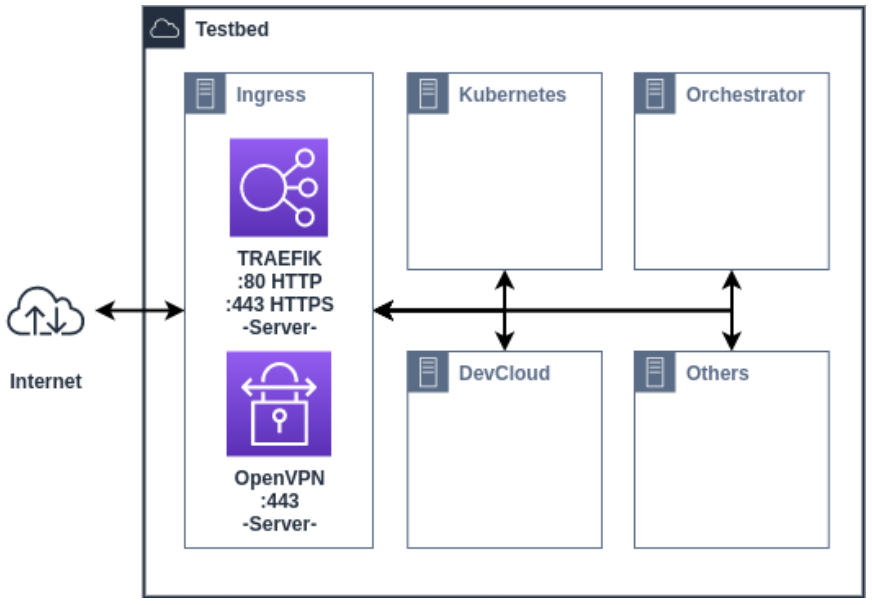

Figure 6 -Development environment, VPN Access

The ingress layer contains a VPN and a single central point of entry for the internal environment services that require public visibility. As a VPN solution, OpenVPN[6] was chosen - a widely-used open source solution for such purpose. In order to discern and encrypt the communications certificates are being used, which encrypt the communication and validate the legitimacy of the VPN public connections. Once the developer is logged into the VPN, it is possible to access all environment resources using the local addresses.

To ensure the security of the environment two main features are applied in the environment. First to block any incoming public request to the environment and secondly to open exceptions for the ingress layer. The ingress layer contains a centralized single point of entry to the resources inside the environment. Therefore any public service that would be in need to have public visibility will have to use this as a point of entry. The solution chosen for this feature is Traefik[7], which is a widely-used open source tool that offers the capabilities such as automatic generation and renewing of SSL certificates for HTTPS traffic, traffic routing and traffic load balancer. In the testbed, Traefik is exposed through HTTP & HTTPS ports. It automatically generates the certificates to enable HTTPS featuring the communication encryption between the clients and the services located inside the environment.

HPC testbed. The intention of the HPC testbed is to provide developers and use case providers with bare-metal compute resources (e.g. CPUs, GPUs) managed by PBS Torque resource manager. This subsection describes the extensions that were provided during this iteration of the testbed.

Figure 7 highlights the current state of the HPC testbed. The testbed was extended to a larger number of nodes (8 compute nodes). This extension now allows to derive a better performance model used by Static Application Optimiser[8].

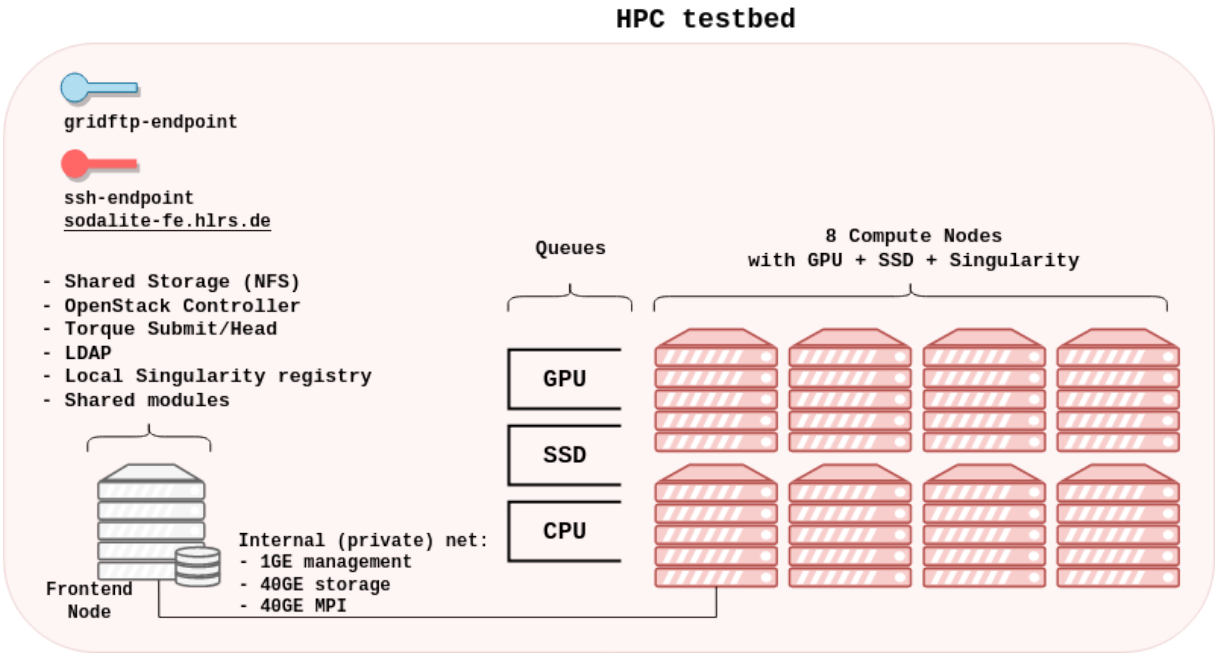

Figure 7 - A functional description of HPC testbed

PBS Torque was configured for GPU scheduling and GPU status monitoring. This allows users of the testbed to directly request GPU resources for job execution as well as to retrieve information about the GPU characteristics and current status. Platform Discovery Service[9] uses this information to enrich resource models of a PBS-compatible target infrastructure with GPU capabilities.

In the real HPC clusters, the usage of queues is common. Queues represent a group of resources with attributes necessary for the queue's jobs. For example, if a job requires a specific resource, such as fast storage, it can be submitted to a specific queue that will schedule the job into the nodes with fast storage. To reflect this, multiple queues were introduced in the HPC testbed to schedule the following resources: GPU for computation acceleration, SSD for I/O acceleration, CPU as a default execution. Platform Discovery Service retrieves information of available queues to create queue capabilities of a resource model for a target cluster.

In order to build and test different variants of optimised containers produced by Static Application Optimiser, various optimisation libraries and compilers were installed (e.g. MPICH, OpenMPI, GCC-9). Additionally, environment modules[10] were introduced to load needed libraries and binaries, resolve possible conflicts and provide needed dependencies.

Finally, a GridFTP server for data transfer was installed. It is extensively used in many HPC centres for reliable, secure and high performance data transfers, also allowing a third-party transfers, i.e. direct cross-server file exchange orchestrated by the client. Therefore, it was important to provide a GridFTP server for the experimentation with data management in HPC.

2.2 Tested Execution Platforms and Infrastructures

SODALITE testbeds provide a testing environment. However, to ensure the SODALITE platform and its constituent components work in production environments, small scale deployments, tests and proof-of-concept on production systems were performed. This section provides the list of tested platforms and infrastructures and the summary is highlighted in Table 3.

A Cray XC40 Piz Daint[11] supercomputer at the Swiss National Supercomputing Centre (CSCS) was used to build and validate the performance model and static application optimisation methodology in general. The evaluation outcome was presented in the deliverable D3.3[12].

The EGI Federation[13] provided SODALITE an access to the virtualized (OpenStack-based federated cloud) and HPC (HPC RIVR[14] - Slovenian supercomputer) resources and "Applications on Demand"[15] services. The deployment of samples and use case components was performed on these resources via SODALITE stack. Additionally, an evaluation of Platform Discovery Service was performed: on one hand, discovering available images, flavors, networks of a federated OpenStack cloud provider and, on the other hand, discovering available HPC resources, such as number of available queues, number of nodes, CPU cores and GPUs.

The deployment on AWS public cloud was also tested. A study on cloud performance[16] was made to compare various cloud providers (EGI, AWS, Azure, GCP) in terms of performance variability.

Table 3 - List of tested production platforms

Infrastructure type	**Infrastructure name**	**Infrastructure description**	**Functionalities tested**
HPC cluster	Cray XC40, Piz Daint	Slurm, 1813 Nodes, CPU Intel Xeon E5-2695 v4 45 216 cores, 84TB RAM	Application optimisation
Federated Cloud	EGI	OpenStack	Deployment Platform discovery Cloud performance
HPC cluster	EGI HPC RIVR	Slurm, 82 Nodes, CPU AMD Epyc Gen 1	Deployment Platform discovery

		4 256 cores, 40 TB RAM	
Public Cloud	AWS	EC2 S3	Deployment Cloud performance

2.3 CI/CD Pipeline

The project chose to preserve its code on the **GitHub** platform (https://github.com/SODALITE-EU), as to facilitate collaboration between the developers of the different components and also to leverage GitHub's excellent Pull Request mechanism, to allow for code reviewing and automatic testing by **Jenkins**.
The **Jenkins** platform was chosen to conduct the CI/CD operations for the project's components, as it is considered one of the most popular CI/CD platforms today and is also available as open-source software. It was decided to run Jenkins and the build slave for it, on top of a **Docker** engine running on a VM inside the project's OpenStack platform.
DockerHub was chosen to store the different **Docker** images produced by Jenkins for the different project components, as the images themselves were deemed to be safe to be deployed to a public registry and **DockerHub** is one of the most popular Docker registries nowadays.
In order to test the quality of the code and to make sure that the Docker images produced were of high quality, it was decided to use the **SonarCloud** platform for static code analysis and code coverage testing.

2.3.1 Project CI/CD setup

During the second year of the project, the project encountered several issues regarding the integration and a more streamlined CD process. The main issues were spotted in these areas:

- **Testbed component deployment**
  - Most components did not deploy automatically to the testbed (prod/staging environments) using CI/CD
- **Docker images of the components**:
  - Images on docker hub had no clear versioning policy
  - Difficult to track the created image in time
- **SODALITE Stack blueprint for deployment**
  - Hard to get a docker image for M18 tag release and produce a working current latest release of the SODALITE deployment blueprint
  - All docker images, used for specific software release (*M18Release*, *M24Release*, *M36Release*) were not properly tagged and used in a tagged version of the SODALITE deployment blueprint
- **Integration tests**
  - Several components did not have test samples and did not provide test coverage for new features

To tackle these issues, the SODALITE Consortium has prepared a convention and is working towards implementing it. It serves as a guide for developers to manage SODALITE releases in a uniform way and its description is outlined in short here.

The convention touches on several areas of development using github and CI/CD pipeline:

- github versioning and usage of tags
- branch naming
- creating scripts for version tag decoding
- setting up a sample jenkins process to cover automated deployment process using xOpera lightweight orchestrator

Strict *semantic versioning*[17] is used to tag versions on github. The unified semantic versioning for project repositories enables a simpler and unified automated deployment mechanism (CD), image label creation for the components and supports better integration testing setups.

Major releases should be tagged accordingly (*M18Release*, *M24Release*, *M36Release*). All other releases must follow Semantic Versioning 2.0.0. SODALITE distinguishes two types of tags: **Pre-release** tags and **Release** tags.

Pre-release tags come in this form: `<major>.<minor>.<patch>-<pre-release>+<build-meta>`, where `<build-meta>` part is optional. Examples:

- 1.1.2-prerelease+meta
- 1.0.0-alpha

Release tags come in these forms: `<major>.<minor>.<patch>+<build-meta>`, where <build-meta> part is optional. Examples:

- 1.1.2+meta
- 1.0.0

Proposed branch schema names

`<type>/<name>`

Type could be any of (feature, feat, fix, bugfix, bug, chore)

Examples:

- `feature/status-endpoint`
- `fix/timeout-bug`
- `chore/CI-CD-upgrade`

Additionally SODALITE uses two GitHub Actions during Continuous Integration to help with later Github Releases: Release Drafter[18] and PR Labeler[19]. The main goal of both tools is to create a draft release that can be easily turned into a real release.

In order for both Actions to work, SODALITE developer should:

- name branches with the proposed brach schema names
- label PRs manually with one of (major, minor, patch)

A sample jenkins file setup was prepared for supporting easier migration to this proposed implementation with several bash scripts for git tag decoding. These parameterized execution of these scripts guides tag identification and decision making for:

- separation of staging docker images and production-ready,
- automatic push on local private or public production docker registry,
- shown the usage of github action tools Release Drafter and PR Labeler for an easier and more streamlined delivery process,
- support for an automated integration test using TOSCA blueprints.

Several of the SODALITE repositories already adhere to this convention and pull requests (PRs) will be created for those who will be considered for this approach. The schema supporting this CI/CD integration is shown in Figure 8.

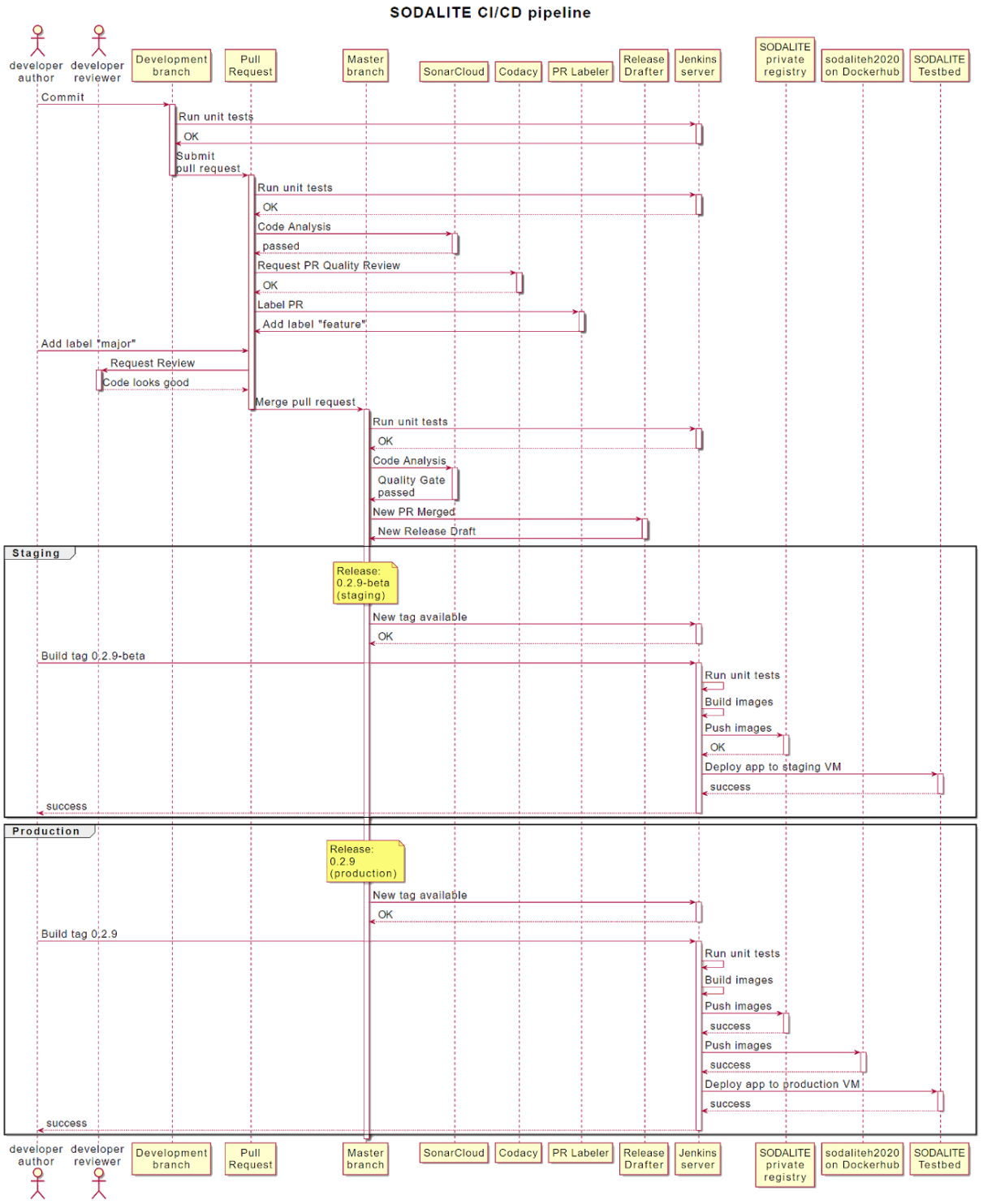

Figure 8 - CI/CD development pipeline

2.3.2 Component integration in the SODALITE blueprint deployment

In order to integrate and deploy SODALITE components as a platform stack, the SODALITE stack TOSCA blueprint[20] was developed. The blueprint contains the topology of the SODALITE stack, i.e. components, dependencies and relationships between the components, and provides a flexible and extendible way to deploy the stack on various targets, e.g., local machine, OpenStack VMs, etc.

Integration tests provide a means to test the SODALITE stack after a new component has been introduced or a component changed version in the SODALITE stack deployment blueprint. After a

component has been added to the blueprint or a functionality changed, the integration tests should be extended to cover this new functionality before merging this version of SODALITE stack blueprint into master.

2.3.3 Example CI/CD Workflow

In order to give a clearer understanding of the CI/CD procedure, we present here details of the CI/CD workflow for one of the SODALITE components, the semantic-reasoner. The example presented here is typical of what happens with most of the SODALITE components. The details are embodied in the Jenkinsfile for the component. We present here highlights from the semantic-reasoner Jenkinsfile, which can be found in its entirety at https://github.com/SODALITE-EU/semantic-reasoner/blob/master/Jenkinsfile.

```
pipeline {
	environment {xxx}
	stages {
		stage ('Pull repo code from github') {xxx}
		stage ('Build the code with Maven') {xxx}
		stage('Sonar analysis') {xxx}
		stage ('Trigger a build of defect-prediction') {xxx}
		stage('Build docker images') {xxx}
		stage('Push Reasoner to DockerHub') {xxx}
		stage('Push graphdb to DockerHub') {xxx}
		stage('Install dependencies') {xxx}
		stage('Deploy to openstack') {xxx}
	}
	post {
		failure {xxx}
		fixed {xxx}

	}
}
```

At the beginning of the Jenkins file a list of environment variables is provided, which include configuration variables that enable us to run the particular job. The heart of the Jenkins process are the stages that are performed. The typical stages for any component are:

- Pull the code from github.
- Build dependencies.
- Build the code and run the unit tests.
- Run the Sonar scanner for static code analysis and code coverage.
- Build other components that depend on the current component.
- If preparing a production run, build the docker images.
- Push the produced images to DockerHub.
- If preparing a production run, perform a full deployment to the OpenStack platform.

A full deployment is performed by Jenkins by running xOpera on the SODALITE stack blueprint, which contains all the relevant SODALITE components.

After the Jenkins process completes, results are reported to the Jenkins dashboard.

The configuration file for the Sonar analysis for the semantic-reasoner looks like this (see https://github.com/SODALITE-EU/semantic-reasoner/blob/master/source%20code/semantic-reasoner/sonar-project.properties):

```
sonar.organization=sodalite-eu
sonar.projectKey=SODALITE-EU_semantic-reasoner
sonar.projectName=semantic-reasoner
sonar.projectVersion=1.0
sonar.branch.name=${BRANCH_NAME}
sonar.sources=.
sonar.java.binaries=.
sonar.coverage.jacoco.xmlReportPaths=./reasoning-engine/target/site/jacoco/jacoco.xml
sonar.inclusions=**/src/**/*.java
sonar.language=java
sonar.sourceEncoding=UTF-8
```

The first few lines simply describe the component to be built. The lines relating to the programming language are used to determine which tools are used to evaluate the code.

Additional details on the development process can be found in Sections 7 and 8 of D2.4[21] (Guidelines for Contributors to the SODALITE Framework).

2.4 Software Quality

The analysis of software quality is executed by means of Jenkins and Sonar, as explained in Section 2.3. More details can also be found in Section 5.2 of deliverable D2.2. Data is pushed to *sonarcloud.io* that stores all the collected KPIs and provides a public dashboard to access them (https://sonarcloud.io/organizations/sodalite-eu/projects).

Table 4 shows the data collected on January 30th 2021 and reports the following metrics:

- Column LOC shows the number of lines of code of each project
- Column QTG reports the last outcome of the analysis process (Quality Gate). OK means passed, KO failed and N/A not available
- Column BUG details the number of code bugs detected by Sonar and the associated label (from best A to worst E)
- Column VLN shows the number of vulnerabilities detected by Sonar and the associated label (from best A to worst E)
- Column HSP reports the percentage of security hotspot reviewed over the total and the associated label (from best A to worst E)
- Column CSL details the number of code smells and the associated label (from best A to worst E)
- Column COV shows the percentage of lines of code covered by test cases
- Column DUP reports the percentage of duplicated lines

Table 4 - Code quality of SODALITE projects

Project Name	LOC	QTG	BUG	VLN	HSP	CSL	COV	DUP
ansible-defects	2200	OK	0 (C)	0 (A)	100% (A)	57 (A)	34.9%	2.9%
application-optimization	4400	OK	0 (A)	0 (A)	- (A)	86 (A)	-	6.2%
hpc-exporter	1100	OK	0 (A)	0 (A)	100% (A)	27 (A)	-	5.9%
edgetpu-exporter	166	OK	0 (A)	0 (A)	100% (A)	0 (A)	-	0.0%

prometheus_ncs2_exporter	196	OK	0 (A)	0 (A)	100% (A)	0 (A)	-	0.0%
iac-blueprint-builder	676	OK	0 (A)	0 (A)	50.0% (C)	14 (A)	87.5%	0.0%
iac-management	144	OK	0 (A)	0 (A)	- (A)	7 (A)	-	0.0%
iac-platform-stack	1	OK	0 (A)	0 (A)	100% (A)	0 (A)	-	0.0%
iac-quality-framework	843	OK	0 (A)	0 (A)	100% (A)	19 (A)	51.8%	0.0%
ide	6600	OK	0 (E)	0 (A)	80% (D)	570 (A)	-	4.7%
image-builder	1800	OK	0 (A)	0 (A)	0.0% (E)	9 (A)	73.6%	1.6%
ipmi-exporter	50	OK	0 (A)	0 (A)	- (A)	1 (A)	-	0.0%
monitoring-system	64	OK	0 (A)	7 (E)	0.0% (E)	0 (A)	-	0.0%
perf-predictor-api	809	OK	0 (A)	0 (A)	100% (A)	59 (A)	34.2%	6.1%
platform-discovery-service	830	OK	0 (A)	0 (A)	0.0% (E)	19 (A)	80,5%	0.0%
refactoring-ct	5400	OK	0 (A)	0 (A)	100% (A)	93 (A)	-	41.5%
refactoring-option-discover	306	OK	0 (A)	0 (A)	100% (A)	20 (A)	90.2%	0.0%
rule-ml-based	1700	OK	0 (A)	0 (A)	100% (A)	76 (A)	98.7%	9.0%
semantic-reasoner	7500	OK	0(A)	0 (A)	100.0% (A)	585 (A)	75.5%	2.9%
ALDE	2000	OK	1 (B)	0 (A)	0.0% (E)	44 (A)	87.5%	0.0%
tosca-smell	971	OK	0 (A)	0 (A)	100% (A)	31 (A)	52.0%	0.0%
verification-unifiedapi	39	OK	0 (A)	0 (A)	100% (A)	0 (A)	-	0.0%
verification-syntax	113	OK	0 (A)	0 (A)	100% (A)	1 (A)	100%	0.0%
verification-workflow	106	OK	0 (A)	0 (A)	100% (A)	1 (A)	22.0%	0.0%
xopera-rest-api	6600	OK	0 (A)	0 (A)	0.0% (E)	52 (A)	80.6%	0.0%

3 Development Status of the MS6 - Second Prototype

This section describes the development status of the MS6 - Second Prototype and its constituent components and modules. Table 5 highlights the overall view on the development, deployment and integration statuses of the SODALITE components done up to Y2 of the project. To elaborate more on the meaning of the statuses, their explanation is provided below:

- Development status means that the source code is released and respective functionally is provided. For Identity and Access Management components, the development status is N/A, since the components were reused as is.
- Deployment status means that the component is deployed via IaC on a particular infrastructure, e.g. Cloud testbed.
- Integration status means that the component is integrated into the platform. Integration is partial if it is integrated with some components either within a layer or across the layers.

During the reporting period, the development of all the components has significantly progressed and new or advanced features have been implemented. New components have been added to further enhance the SODALITE platform, such as MODAK and Platform Discovery Service, described in detail in D4.2. Furthermore, new Identity and Access Management (IAM) components were introduced into the overall SODALITE architecture, as presented in D2.2. The integration of tools for Software Quality measurement helped to continuously estimate the quality of the developments.

The integration work was significantly improved through the improved CI/CD agreed convention for release management and an introduction of SODALITE IaC Platform Stack[22]. New components are partially integrated, such as MODAK, Platform Discovery and IAM.

Table 5 - Development status of the MS6 - Second Prototype

Semantic Modelling Layer			
Component	**Development**	**Deployment**	**Integration**
SODALITE IDE			
Semantic Reasoner			
Semantic KB			
IaC Management Layer			
Component	**Development**	**Deployment**	**Integration**
Abstract Model Parser			
IaC Blueprint Builder			
Runtime Image Builder			
Concrete Image Builder			
Application Optimiser - MODAK			
IaC Verifier			
Verification Model Builder			
Topology Verifier			
Provisioning Workflow Verifier			
Bug Predictor and Fixer			
Predictive Model Builder			
IaC Quality Assessor			
IaC Model Repository			
Image Registry			
Platform Discovery Service			

Runtime Layer			
Component	**Development**	**Deployment**	**Integration**
Orchestrator + Drivers			
xOpera REST API			
Monitoring			
Deployment Refactorer			
Node Manager			
Refactoring Option Discoverer			
Kubernetes Edge Components			
Identity and Access Management Components			
Component	**Development**	**Deployment**	**Integration**
IAM Introspection	N/A		
Secrets Management	N/A		
Table legend	**Completed**	**Partial**	**Not started**

A detailed description of the development and integration status of each layer of the MS6 - Second Prototype is outlined in D6.6 Appendix A, additionally presenting the location to the source code and downloadable artifacts, dependencies and steps towards the next prototype.

4 Development Status of the Demonstrating Use Cases

This section provides the development status of the three demonstrating use cases of SODALITE: POLIMI Snow, USTUTT Virtual Clinical Trial and ADPT Vehicle IoT.

4.1 POLIMI Snow UC

The goal of this use case is to exploit the operational value of information derived from public web media content, specifically from mountain images contributed by users and existing webcams, to support environmental decision making in a snow-dominated context. An automatic system crawls geo-located images from heterogeneous sources at scale, checks the presence of mountains in each photo and extracts a snow mask from the portion of the image denoting a mountain.
Two main image sources are used: touristic webcams in the Alpine area and geo-tagged user-generated mountain photos in Flickr in a 300 x 160 km Alpine region. Figure 9 shows the different components of the pipeline, all of which were developed.

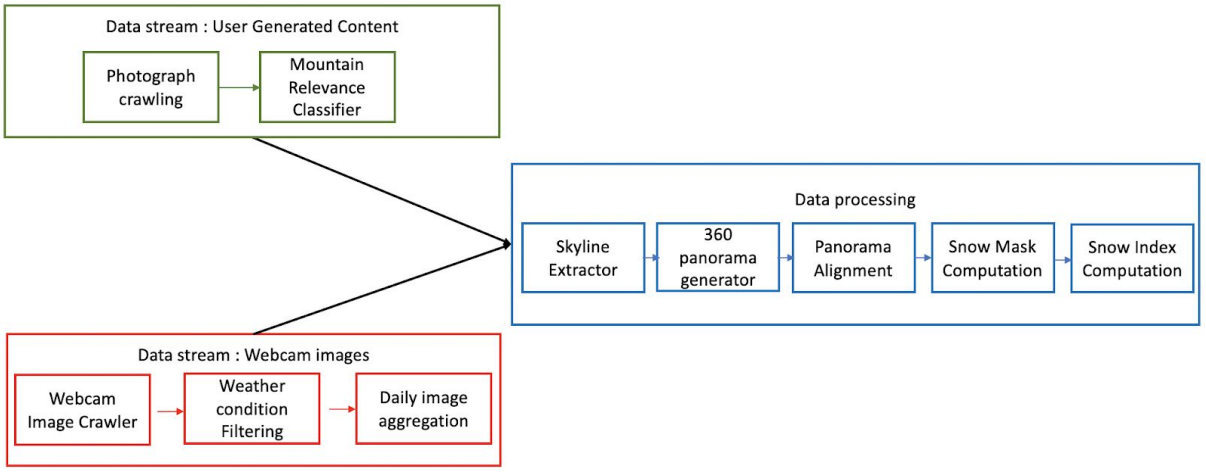

Figure 9 - Components of the Snow Use Case pipeline

Our work is based on the implementation plan presented in deliverables D6.1[23] and D6.2[24] SODALITE Platform and Use Case Implementation Plan. In Figure 10 we present the planned schedule with the released components highlighted. All components planned for the Y2 were developed according to the schedule:

Delivered in Y1	Delivered in Y2	Expected in Y3
Webcam image crawler (WIC)	User generated image crawler (UGIC)	No new components are expected
Weather condition filter (WCF)	Mountain relevance classifier (MRC)	
Daily median image aggregation (DMIA)	Snow mask computation (SMC)	
Skyline extraction (SE)	Snow index computation (SIC)	
360° panorama generation		

(360PG)		
Peak alignment (PA)		

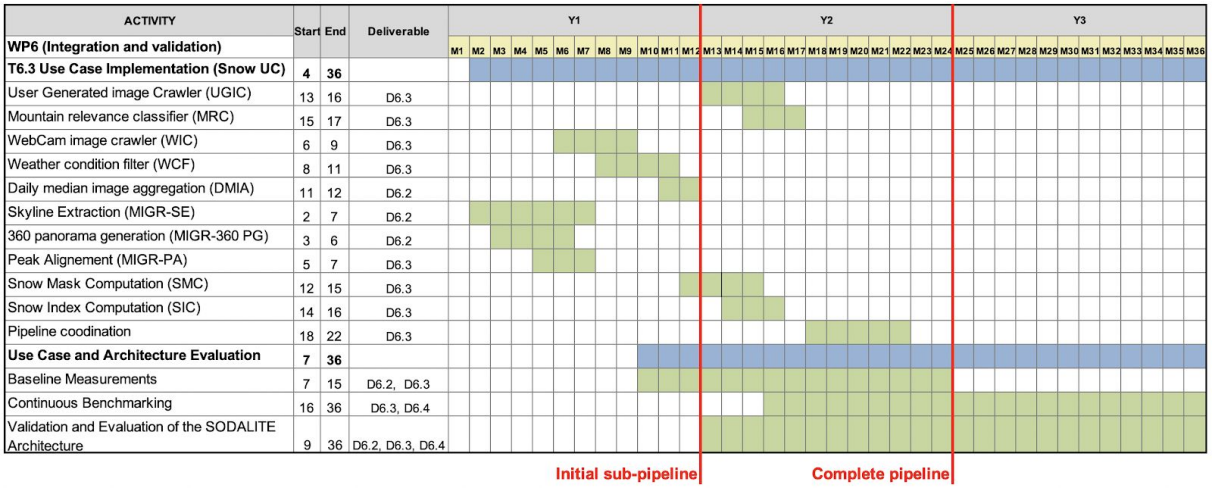

Figure 10 - Implementation Plan of Snow use case

A review of the developed components, which are being deployed in the testbeds, is presented in the subsequent sections.

4.1.1. User Generated Image Crawler (UGIC)

Flickr is selected as the data source for user-generated photographs, because it contains a large number of publicly available images, many of which have an associated geotag (GPS latitude and longitude position saved in the EXIF (Exchangeable Image File Format) container of the photograph).

The Flickr API allows one to query the service using temporal and spatial filters. A user generated images (UGI) crawler algorithm is designed to query sub-regions on the area of the Alps.

Table 6, provides a summary of the user generated image crawler (UGIC) component.

Table 6 - Webcam image crawler component summary

Input	Coordinates of the search region bounding box Mountain-related textual keywords
Processing	• Opens a connection to the query API of the user-generated image repository • Submits queries formulated with the input keywords and area • Retrieves images that match query and save their reference • Given the images and references, it downloads the images, and save data in database
Output	Images are saved on disk and in database

Implementation technologies and languages	• JAVA • MySQL

Examples of crawled images are shown in Figure 11.

Figure 11 - Crawled images examples

4.1.2. Mountain relevance classifier (MRC)

Pictures tagged with a location corresponding to a certain mountainous region do not ensure the presence of mountains. For this reason, the presence of mountains in every photograph is estimated and the non-relevant photographs are discarded. The process to classify an image first computes a fixed-dimensional feature vector, which summarizes the visual content, and then provides it to a Multi-layer Perceptron (MLP) classifier to determine whether the image should be discarded or not. A dataset of images annotated with mountain/no mountain labels is needed to train the model.
Examples are shown in Figure 12 and Table 7 summarizes the mountain relevance classifier (MRC) component.

Figure 12 - Crawled image of a mountain (left) and not a mountain(right)

Table 7 - Mountain relevance classifier summary

Input	An image
Processing	• Calculate Image Features • Input features into the Multi-layer Perceptron classifier • Calculate the probability of the image to correspond to a mountain picture and based on a threshold returns the value

Output	Value indicating if the image corresponds to a mountain image or not
Implementation technologies and languages	• Python

4.1.3. Snow mask computation (SMC)

A snow mask is defined as the output of a pixel-level binary classifier that, given an image and a mask M that represents the mountain area as inputs, produces a mask S that assigns each pixel of the mountain area a binary label denoting the presence of snow. Snow masks are computed using the Random Forest supervised learning classifier with spatio-temporal median smoothing of the output. To perform the supervised learning a dataset of images with an annotation at pixel level indicating if the pixel corresponds to the snow area is needed.
The snow mask computation component (SMC) is described in Table 8.

Table 8 - Snow mask computation component summary

Input	An image and a mask indicating the pixels corresponding to the mountain area.
Processing	• Calculate feature vectors for the pixels in the mountain area • Input the features into the Random Forest Classifier
Output	S = Snow mask indicating for each pixel if it represents snow or not in the original image.
Implementation technologies and languages	• Python

Examples are shown in Figure 13.

Figure 13 - Example of an image and its snow mask generated

4.1.4. Snow index computation (SCI)

The pipeline produces a pixel-wise snow cover estimation from images, along with a GPS position, camera orientation, and mountain peak alignment. Thanks to the image geo-registration and orthorectification (using the associated topography data) it is possible to estimate the geographical properties of every pixel, such as its corresponding terrain area and altitude.

Consequently, it is possible to compute the snow line altitude (the point above which snow and ice cover the ground) expressed in meters.
The virtual snow index for an image is defined as: $\Sigma_{(x,y)\,|\,S(x,y)=1}\, vsi(x,y)$, where vsi is a virtual snow index function that transforms a pixel position into a snow relevance coefficient and can be defined as $vsi_{(x,y)} = 1$ and $S(x,y) = 1$ indicates it will be calculated for each pixel that corresponds to the snow mask obtained in the previous step.
In Table 9, a summary of snow index computation(SIC) component is presented.

Table 9 - Snow index computation component summary

Input	S = the snow mask M = the mountain area mask
Processing	• Filter pixels in the mountain mask area • Calculates the VSI • Calculate percentage of snow on the image based on the mountain mask area
Output	Virtual snow index, Snow percentage
Implementation technologies and languages	• Python

4.2 USTUTT Virtual Clinical Trial UC

The “In-silico clinical trials for spinal operations” use case targets the development of a simulation process chain supporting in-silico clinical trials of bone-implant-systems in Neurosurgery, Orthopedics and Osteosynthesis. It deals with the analysis and assessment of screw-rod fixation systems for instrumented mono- and bi-segmental fusion of the lumbar spine by means of continuum mechanical simulation methods.

The initial layout of the process chain was described in the deliverable D6.1 (Section 4.2). After the detailed analyses of the clinical imaging Data, whose results are presented in D6.2 (Section 4.2), it became obvious that additional image processing steps are necessary to ensure proper image quality. The current layout of the process chain as described in D6.2 is depicted below in Figure 14.

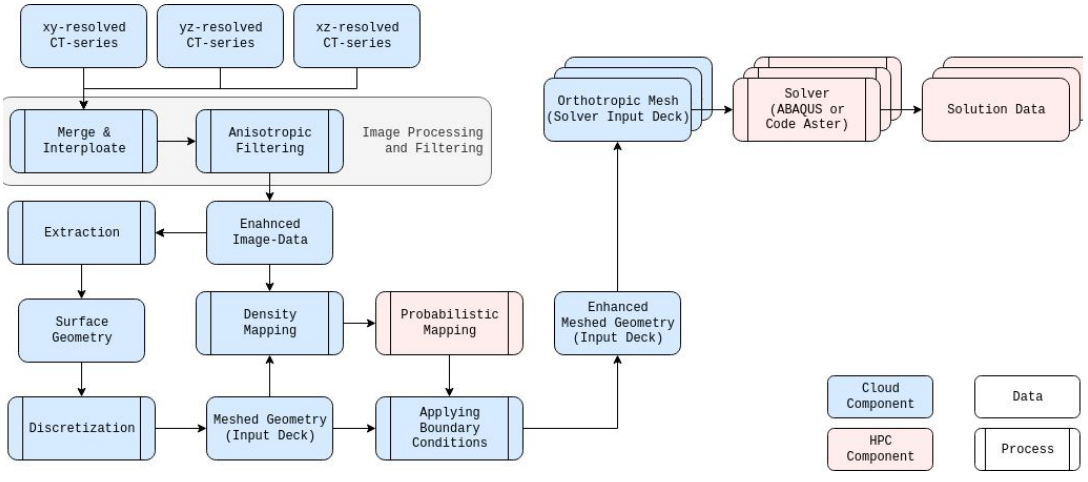

Figure 14 - Schema of the Virtual Clinical Trial use case pipeline with additional steps

Our work on the process chain is based on the implementation plan presented in deliverables D6.1 and the developments presented in D6.2. In Figure 15 we present the planned schedule. All components planned for the Y2 were developed according to the schedule even though the implementation of the Final UC Process is currently delayed due to the introduction of additional image processing components.

Delivered in Y1	**Delivered in Y2**	**Expected in Y3**
Extraction	Image Processing and Filtering	Process integration
Discretization	Boundary Condition	No new components are expected
Density Mapping	Solver	
Probabilistic Mapping		

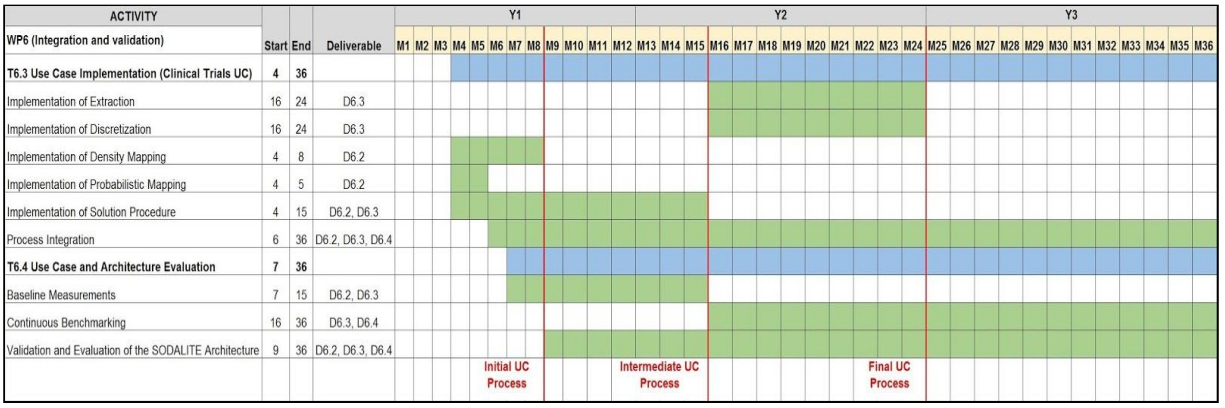

Figure 15: Gantt diagram of the development timeline of the Virtual Clinical Trial use case components

In the subsequent sections an overview about the Y2 developments is presented.

4.2.1 Image Processing and Filtering

Since the image processing and filtering component is composed out of several processing steps, our choice was to implement it as an integrated pipeline based on the Visualization Toolkit[25] (VTK) which provides all steps as ready to use algorithms with corresponding python bindings. As input the implemented pipeline takes three image series in VTK file format[26], which are reconstructed in different image planes i.e. whose resolution differs along the three coordinate axes as described in D6.2. On output it delivers an integrated dataset in VTK file format.

Since the data are represented as image data, i.e. as a rectilinear grid, each dataset's reconstruction plane is originally aligned along the xy-coordinate plane. Due to that the first step is to rotate them back into their original scanner coordinate system as specified in the header of the original DICOM image series. Subsequently all three datasets are resampled and interpolated to a rectilinear grid which provides high resolution in each of the three coordinate planes.

In Table 10, a summary of the Image Processing and Filtering component is presented.

Table 10 - Image Processing and Filtering component summary

Input	• Three data sets reconstructed in different image planes • One coordinate system per dataset
Processing	• Back-transformation of each input dataset into the original scanner coordinate system • Resampling of each dataset to a high-resolution rectilinear grid • Fusion and filtering of the three data-fields
Internal concurrency	No, sequential process
Output	High resolution Image dataset
Implementation technologies and languages	• VTK • Python

4.2.2 Applying Boundary Conditions

The "Applying Boundary Conditions" component is derived from the original implementation of the "Density Mapping" component. The component integrates each of the three output files of the "Probabilistic Mapping" component with the "Meshed Geometry" in Code_Aster med file format. It generates three "Enhanced Meshed Geometries" in Code_Aster's med file format which include inhomogeneously distributed material Information needed by Code_Aster. Due to the development of the Image processing and filtering component the output is currently not fully completed which is the last link missing to fully connect the processing pipeline.

In Table 11, a summary of the Applying boundary component is presented.

Table 11 - Applying boundary component summary

Input	• One input deck with meshed geometry. • Three data files containing high, low and mode density distributions
Processing	From the high, low and mode density distributions a high low and mode material is generated per element inside the meshed geometry
Internal concurrency	No, sequential process
Output	Three modified solver input decks per patient.
Implementation technologies and languages	Fortran

4.2.3 Solver

As described in D6.2 the transfer of the model to the Code_Aster solver was already done even though execution of the component was only possible sequentially. During year 2 the model execution in parallel was set up which is now possible by means of the direct solver MUMPS as well as the iterative solver library PETSc. Also parallel mesh processing is possible by means of the graph partitioner METIS. During the setup of parallel Code_Aster it became obvious that the build process depends on several external libraries which are: (1) a sequential Code_Aster installation, (2) ptscotch[27], (3) parmetis[28], (4) OpenBLAS[29], (5) lapack[30], (6) sclapack[31], (7) blacs[32], (8) MUMPS[33], (9) superlu[34], (10) PETSc[35] and (11) libmed[36].

To enable the deployment of parallel Code_Aster on the SODALITE infrastructure the build sequence of the dependencies was documented. To enable tests for automatic performance optimization a scalable test model using all features of the final biomechanical models was build and bare metal runtime tests on the HLRS system Vulcan were carried out using computing nodes with Intel Xeon Gold 6138 @ 2.0GHz Skylake processors and 192 Gb of main memory. The results of the runtime tests are given in Table 12.

Table 12 - Results of bare metal runtime tests of parallel Code_Aster

Solver	Partitioner	#-MPI-Procs	Walltime [s]
MUMPS	-	4	654.52
MUMPS	-	8	442.66
MUMPS	METIS	8	226.87
PETSc	METIS	8	67.43

In Table 13, a summary of the Solver component is presented.

Table 13 - Solver component summary

Input	Enhanced solver input decks per patient.
Processing	Using finite element methods, a solution is computed for lower and upper bound of HDI as well as for the mode. These three solutions are computed.
Internal concurrency	MPI can be used for parallel computation. OpenMP can be used with the PETSc solver library
Output	For each input deck a solution file is computed.
Implementation technologies and languages	• Sequential Code_Aster • ptscotch • parmetis • OpenBLAS • lapack • sclapack • blacs • MUMPS • superlu • PETSc • libmed

4.3 ADPT Vehicle IoT UC

The Vehicle IoT use case focuses on situationally-aware processing of data subject to various latency, security, and regulatory constraints within a connected vehicle. The precise requirements of the workload are subject to change based on factors such as the regulatory environment, the privacy preferences of the driver, resource availability, requisite processing power, connectivity state, etc. This use case targets mixed Cloud/Edge deployment models and focuses on dynamic

adaptation and run-time redeployment/reconfiguration in order to satisfy both its performance and compliance requirements.

The implementation plan for the UC is outlined in Figure 16 below:

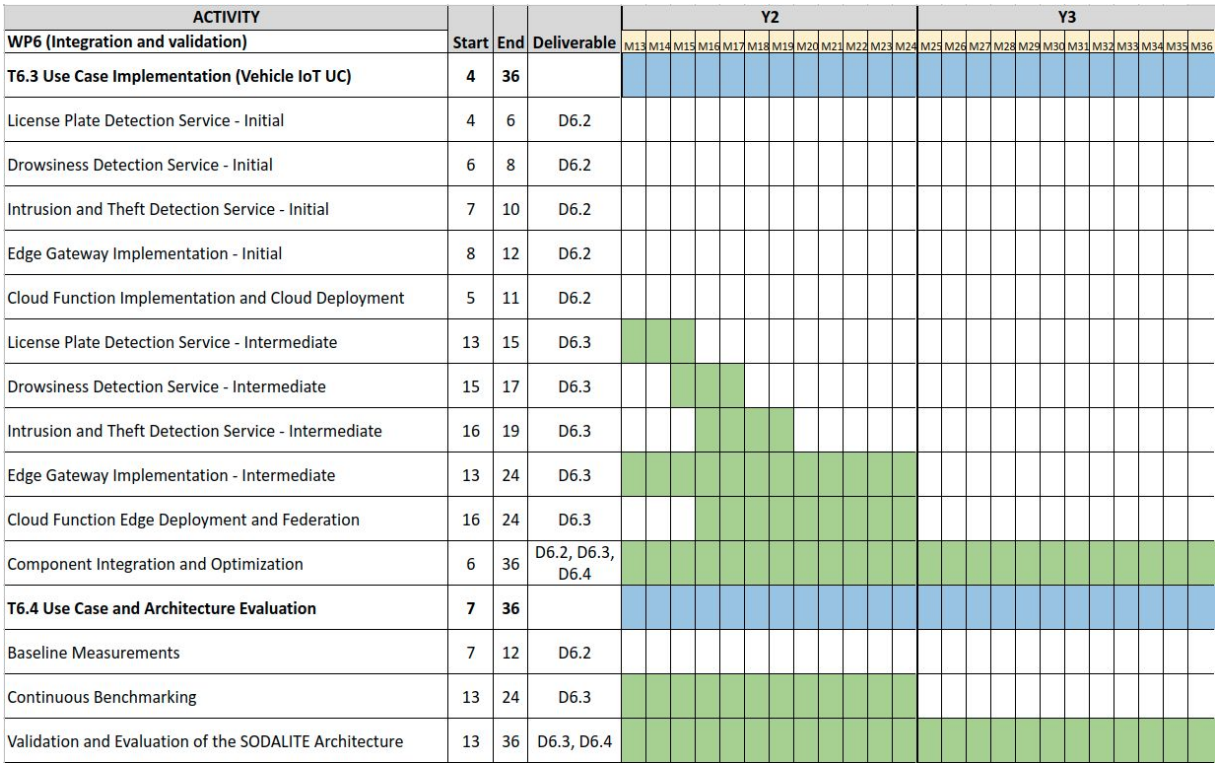

Figure 16 - Vehicle IoT UC Implementation Plan

At present, all of the planned use-case-specific development work for Y2 work has been carried out on schedule. As the use case requires management and orchestration of an Edge-based Kubernetes cluster by the SODALITE stack, and many of the requisite components will not provide Kubernetes support before the end of Y2, there has been slippage of the continuous benchmarking task. Continuous benchmark has, therefore, been deferred to Y3, and will be carried out alongside the integration and validation task. Cloud functions have been used experimentally within the context of the use case itself, but any further work on this remains similarly blocked until there is better integration between SODALITE and the Edge. This work is similarly expected to be completed in Y3. A brief summary of components and their respective features at different points in the project timeline is provided in the table below:

Table 14 - Vehicle IoT UC component/feature timeline

Component	Y1 Features	Y2 Features	Y3 Features
License Plate Detection Service	Initial implementation of ALPR microservice.	Edge-based triggering through OpenFaaS Cloud-based OCR model retraining.	Leveraging Cloud/HPC for online model retraining and Edge delivery (use of accelerators, where possible).

			Re-training pipeline through FaaS/MLops.
Driver Monitoring Service	Initial implementation of drowsiness detection model.	Extension for multithreading, additional monitoring & alerting models.	Dynamic use of heterogeneous accelerators to improve FPS.
Intrusion and Theft Detection Service	Initial implementation of intrusion and theft detection service. Able to detect pre-enrolled facial recognition models and generate alerts.	Extension for enrolling new users, training new recognition classifiers.	Leveraging Cloud/HPC for online model retraining and Edge delivery (use of accelerators, where possible). Re-training pipeline through FaaS/MLops.
Edge Gateway	Edge Gateway provisioning, ability to run limited backend components.	Edge Gateway as managed Kubernetes node. Kubernetes-based feature discovery, node labelling and monitoring.	Dynamic refactoring of per-node deployments based on resource changes, metrics.
Edge Exporter	Application and Edge node exporters.	Metric exporters for heterogeneous accelerators (GPU, NCS2, EdgeTPU) Static alerting rules for refactoring.	Dynamic alerting rules based on specific Edge Gateway configuration.

The Y2 work has focused largely on exposing Edge-based metrics and capabilities to the SODALITE run-time monitor, as well as tying Edge-based alerts into the refactorer in order to update the deployment in response to environmental alerts (e.g., exceeding thermal tolerations). While the Y2 work has shown that SODALITE is capable of limited reconfiguration of Edge-based deployments from the Cloud, further integration between the knowledge base, platform discovery service, and the Edge-based Kubernetes nodes is required in Y3.

4.3.1 Edge Gateway

During Y1, the main focus of the Edge Gateway implementation was to provide in-vehicle instantiation of basic vehicle services that at the start of the project were only deployed in Cloud. The work in Y2 has focused primarily on: (1) providing a clearer separation between services that can, or must, be deployed directly at the Edge, and ones that are able to run in Cloud; (2) increasing service sophistication to take advantage of Edge-local processing; and (3) exposing the unique capabilities and operating environment of each Edge Gateway to the SODALITE stack in order to lay the groundwork for SODALITE-driven optimization and deployment reconfiguration in Y3.

A key challenge for SODALITE is dealing with dynamism not just in the deployed applications that are likely to be used within different vehicles (further detailed in the following sections), but also in variance in the Edge Gateway implementation itself. To this extent, and as outlined in Table 2 above, a number of different hardware configurations with different heterogeneous accelerators

are being utilized in the Edge testbed to provide an environment representative of what would typically be found in an ad-hoc or managed vehicle fleet involving different types of vehicles at different stages of their service lifetime. Addressing these heterogeneity challenges in a simplified way is a key requirement to moving towards a more hierarchical deployment model through the introduction of an additional Fleet Gateway in Y3. The current and planned deployment phase for the use case is exemplified in Figure 17 below:

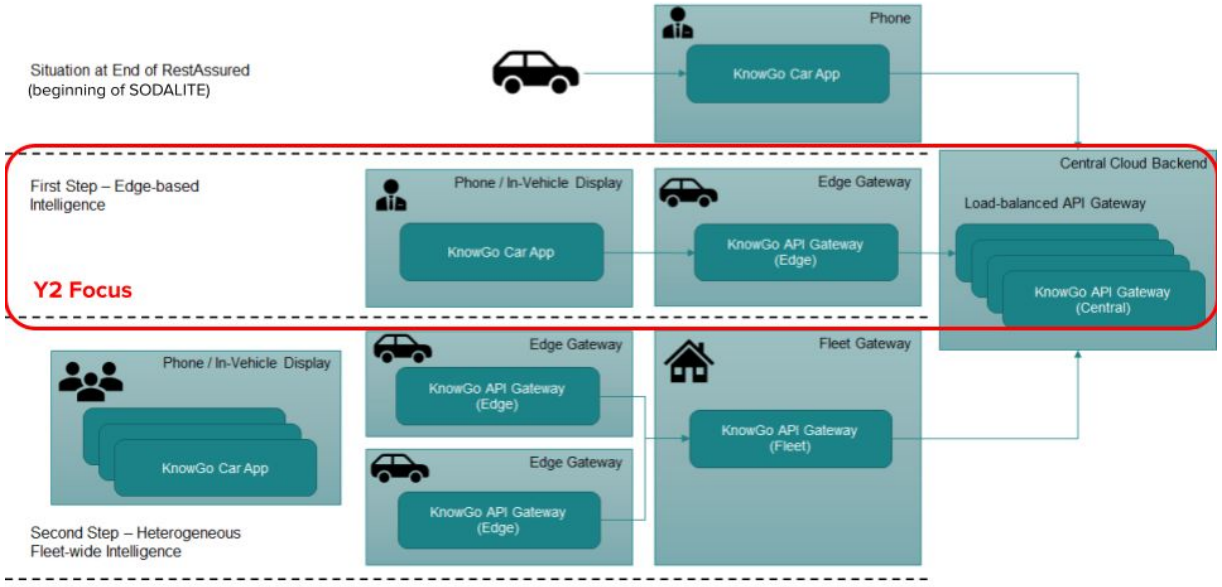

Figure 17 - Vehicle IoT UC deployment phases

In order to close the gap to production at the end of the project, the Y2 deployment phase has focused primarily on establishing Cloud-to-Vehicle provisioning and service deployment within the context of a Kubernetes cluster (representative of production deployment), and in ensuring that SODALITE is able to carry out simple provisioning and refactoring of Cloud-to-Edge deployments within Kubernetes.

In this case, each Vehicle contains its own Edge Gateway, managed as a Kubernetes Node. This may be managed directly from Cloud to Vehicle, or in the fleet management case, hierarchically, with each Fleet Gateway acting as the Kubernetes master node for a cluster of vehicles, with the cloud acting as a federation controller across fleet clusters. The different types of Kubernetes deployment scenarios are exemplified below:

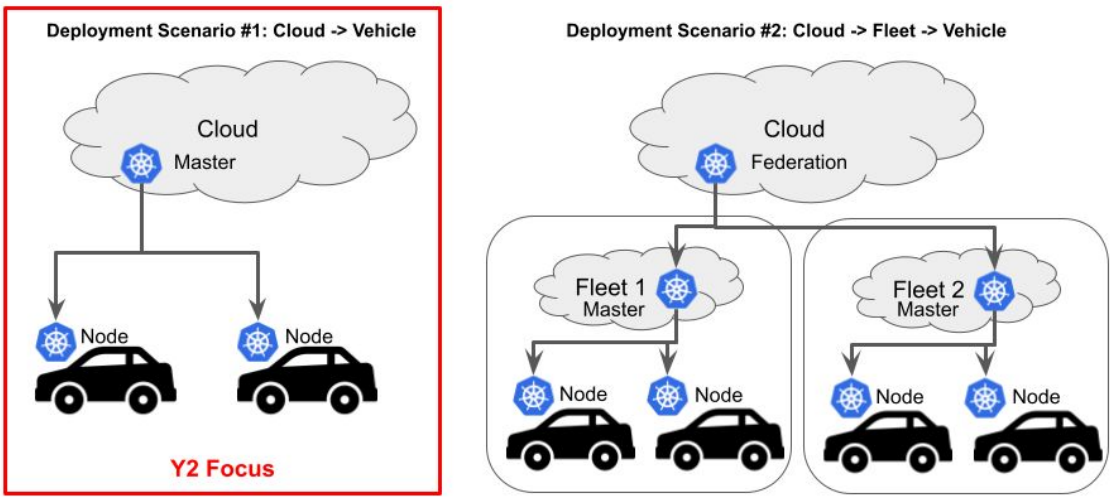

Figure 18 - Vehicle IoT UC Kubernetes Deployment Scenarios

In Y3, it is planned to expand the Y2 work for cross-cluster federation, in order to enable vehicles to be grouped into self-contained vehicle fleets, while permitting vehicle services and monitoring to be provided at both the fleet and cloud levels.

4.3.2 Vehicle Services

A number of vehicle services, dedicated microservices taken from ADPT's KnowGo Car[37] platform, are leveraged within this use case. These are briefly expanded upon in the subsections below. During Y1, these services were constrained to the Cloud backend, while progress in Y2 has seen them pushed down to the in-vehicle Edge Gateway, where they have been further refined and integrated with SODALITE components.

4.3.2.1 Driver Monitoring & Alerting Service

The driver monitoring service implements a meta model for driver alertness monitoring and alerting based on real-time analysis of an in-vehicle video stream. Driver alertness is determined from a number of different variables, with per-detector weighting adjusted to reflect model confidence. Based on the weighted average of the detection results, the degree of alertness is determined, and a proportionate alert triggering an in-vehicle actuation event can be generated (e.g. a minor level of inattentiveness may trigger an auditory warning, while more serious cases may trigger force-feedback events, with the amplitude and magnitude of the sinusoidal wave generating the vibration pattern scaled in proportion to the level of detected risk). A number of detection models used by the service are outlined in the table below:

Table 15 - Detection models used by the Vehicle IoT driver monitoring service

Detection Model	Model Description	Implementation Technologies
PerclosDetector	Measure drowsiness based on the percentage of eyelid closure	OpenCV/Dlib
YawnDetector	Measure drowsiness based on the detection of yawns, counting frequency and duration	PyTorch
GazeDetector	Measure the extent to which the driver is	Tensorflow/Keras

	watching the road through pose estimation	

In the Y1 implementation, all analysis was carried out on the Cloud side, with video being streamed from the vehicle to the Cloud. This has been suboptimal for a number of reasons:

1. The amount of bandwidth required to stream the video;
2. The per-stream computation overhead limiting the number of streams that can be analyzed in parallel without incurring disproportionate expense on the Cloud side;
3. Safety risks owing to the round-trip latency between detection and alerting, particularly when network connectivity is poor; and
4. Low social acceptance by drivers, who are less inclined to enable the service when there is an externally accessible video stream into their vehicle.

At the end of Y1, in-Cloud video analysis averaged between 5-6fps on an unaccelerated CPU-based deployment, with a camera providing imagery at 30fps.

Progress in Y2 has enabled this service to be deployed locally within the vehicle onto the Edge Gateway, allowing the service to directly consume and process the video stream at the Edge, and to only pass on summary statistics through a JSON payload to external services (both at the Edge and in the Cloud) consented to by the driver.

The service itself can be deployed directly from the Cloud to the Edge by the SODALITE orchestrator, and Edge-based exporters are able to provide metrics concerning the application performance, as well as the thermal characteristics of both heterogeneous accelerators and the Edge Gateway itself. Preliminary experimentation at M18 has already demonstrated that the SODALITE refactorer, informed by the run-time monitor, is able to act on pre-defined thermal-based alerting in order to scale the deployment directly in the Kubernetes cluster. An overview of the interactions between SODALITE and the driver monitoring service is provided in Figure 19 below:

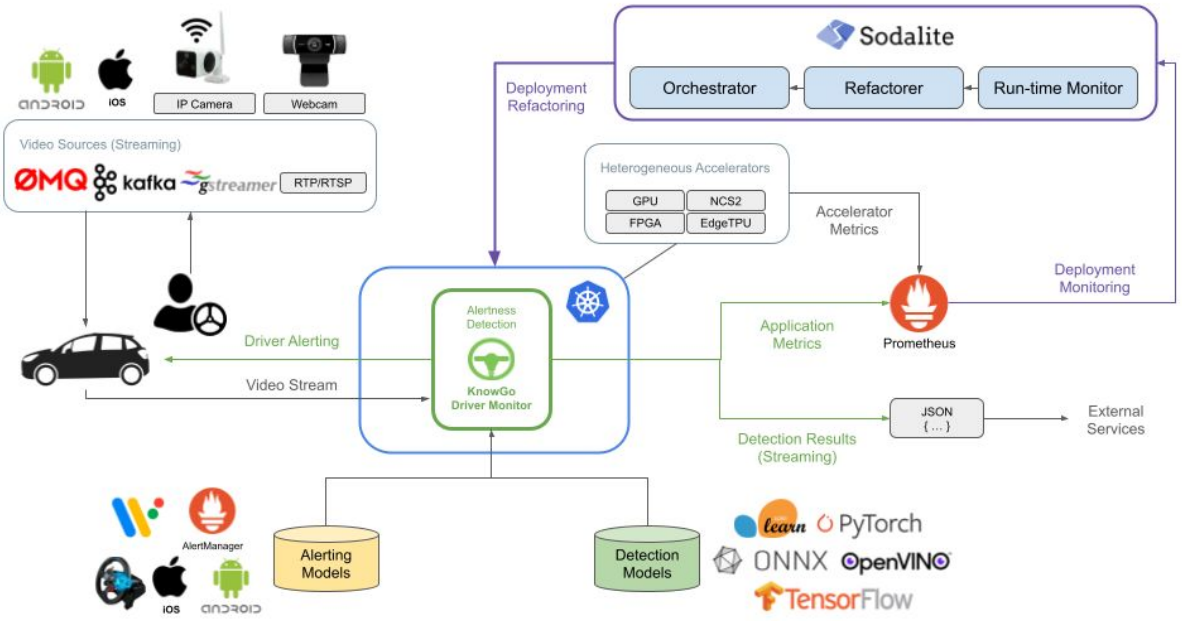

Figure 19 - Driver monitoring service & SODALITE interaction

At the end of Y2, video analysis can be carried out at an average of 12fps on a relatively low-powered Edge device on an unaccelerated CPU-based deployment.

The use of heterogeneous accelerator resources is currently carried out statically within the service-specific Kubernetes manifest, limiting the option for resource sharing between services. In this case, both the driver monitoring service and the risk scoring service that the monitoring service integrates with can benefit from GPU acceleration, while in practice only one can use the GPU at a time. The monitoring service, furthermore, due to its need to process video frames as close to real-time as possible, has a greater need of GPU-based acceleration than the scoring service, where it is simply nice to have, if available. These aspects will be addressed in Y3.

The focus for Y3 will, therefore, be in the following areas:

1. Online optimization of the deployed application in response to changes in accelerator availability and environmental conditions;
2. Determining optimal deployment blueprints based on the deployed applications, the available resources, and the importance each service attributes to a specific accelerator type; and
3. Enabling continuous delivery of improved AI/ML models from the Cloud to the Edge, adapted for a range of heterogeneous accelerators that may be encountered in the Edge Gateway.

4.3.2.2 Intrusion and Theft Detection Service

The intrusion and theft detection service provides a driver identification and authorization flow based on analysis of in-vehicle camera imagery.

At the end of Y2, the service has been successfully deployed at the Edge, and is able to run standalone or in conjunction with the driver monitoring service (in the latter case, the video stream is shared between the services, until driver identification is carried out and authorization is granted). The service may be periodically invoked (e.g. upon token expiry, or in changes to the ignition status when a new journey is commenced) in order to re-authorize the driver. Fleet managers (or vehicle owners) are able to dynamically add/remove authorized drivers from the Cloud (or via their driver app) to a given vehicle, and are able to receive alerts upon authorization failure. Identification results are also signalled to the risk scoring service, allowing the overall risk level to be increased/reduced based on operator-defined weighting.

The service is also able to benefit from periodically improved driver identification models, particularly when a driver has only just begun using the service and the overall prediction confidence is low. Model accuracy improvements also have benefits for the driver beyond the reduced risk of false negatives - the higher degree of confidence with which a driver can be identified, the more the driving risk score can be reduced. While progress in Y2 has provided a mechanism by which model re-training can be initiated in-place, more work is required in order to facilitate Cloud-to-Edge model delivery with transfer learning. The Y3 work is expected to look at improving this, and will reuse the work done in the driver monitoring service in order to construct a SODALITE-driven MLOps pipeline.

While the service runs as a dedicated microservice, the interactions between services are presently tightly coupled. As there are multiple flows in which driver identification and authorization should be carried out, and the service only needs to be periodically invoked, this is seen as an ideal candidate for FaaS-based triggering, which will be further explored in Y3.

5 Validation and Evaluation of the MS6 - Second Prototype

Introduction

This section covers the evaluation of the second prototype of SODALITE, considered both from the point of view of the SODALITE platform itself, as well as the application by demonstrating use cases. In the following subsections, validation and evaluation is broken down by focus area:

- **Use case validation**: This subsection looks at the current coverage of SODALITE UML UCs from the point of view of each demonstrating use case, the improvement in uptake of UML UCs YoY, and a detailed look at how these are being applied to provide value for each of the demonstrating use cases.

Evaluation of the SODALITE platform:

- **Modelling abstractions**: This subsection looks at the extent to which the modelling layer is able to model the infrastructure and application patterns, infrastructure performance characteristics, and execution constraints of each of the demonstrating use cases.
- **Performance**: This subsection looks at the efficacy of MODAK in performing static optimizations, as well as the application of refactoring and run-time reconfiguration in order to mitigate SLA violations.
- **Usability**: This subsection assesses the overall usability and perceived usefulness of the SODALITE IDE. Controlled experiments have been run with use case owners and with TOSCA experts to determine how users with different levels of expertise are able to make effective use of the SODALITE IDE in application modelling.
- **Integration KPIs**: This subsection looks at the extent of component integration achieved, while also assessing the extent to which SODALITE outputs have been released as open source, and where these outputs have been contributed back to upstream open source projects.

This section finally concludes with an evaluation summary, providing a consolidated view of the achieved KPIs, a summary of the overall evaluation, and the identification of outstanding issues to be addressed in Y3.

5.1 Use case validation

All three demonstrating use case owners have validated the SODALITE platform (MS6 - Second Prototype), covering a greater number of SODALITE UML UCs, as shown in Table 16. In the following subsections, each demonstrating use case presents its validation scenarios and elaborates more on the coverage of UML UCs, as well as reports on the improvements achieved when using SODALITE platform.

Table 16 - Coverage of the SODALITE UML use cases by the demonstrating use cases by M24

Use Case	Virtual clinical trial	SNOW	Vehicle IoT	Testbed Providers
UC1 Define Application Deployment Model (WP3)				
UC2 Select Resources (WP3)				
UC3 Generate IaC code (WP4)				
UC4 Verify IaC (WP4)				
UC5 Predict and Correct Bugs (WP4)				
UC6 Execute Provisioning, Deployment and Configuration (WP5)				
UC7 Start Application (WP5)				

UC8 Monitor Runtime (WP5)				
UC9 Identify Refactoring Options (WP5)				
UC10 Execute Partial Redeployment (WP5)				
UC11 Define IaC Bugs Taxonomy (WP4)				
UC12 Map Resources and Optimisations (WP3)				
UC13 Model Resources (WP3)				
UC14 Estimate Quality Characteristics of Applications and Workload (WP3)				
UC15 Statically Optimize Application and Deployment (WP4)				
UC16 Build Runtime images (WP4)				
UC17 Platform Resource Discovery (WP4)				

Y1	Y2

Concerning UC14, this is expected to be used with the demonstrating use cases in Y3. While some initial performance modelling of the use cases has been done with MODAK in Y2, the bulk of this activity is not planned before the last year as the focus shifts towards increased integration and optimization.

5.1.1 Snow UC

The goal of this use case is to exploit the operational value of information derived from public web media content, specifically from mountain images contributed by users and existing webcams, to support environmental decision making in a snow-dominated context.

Environment intelligence requires sophisticated data acquisition and analysis pipelines, which must scale to large volumes of data, deliver predictions and alerts with severe time constraints. SODALITE optimisation and reconfiguration of the architecture improves detection and prediction accuracy due to improved throughput of data. SODALITE enables a simplified deployment and management of data intensive pipelines capable of processing visual data at scale and with a high throughput by reducing the complexity of the operations and making them manageable by non highly skilled IT specialists.

In this context, SODALITE can streamline the design, configuration deployment and monitoring of the infrastructure underlying any such environment intelligence pipeline, by alleviating the technical skill level necessary to administer such a complex architecture and thus lowering the entry barriers for public administration and utility companies wishing to reap the benefits of advanced computer vision and deep learning solutions for optimizing their operations. Specifically, the ability to conceptually model the pipeline architecture and generate the deployment artifacts is an essential factor for shortening the deployment time of many environment intelligence applications.

During the first year we delivered a sub-pipeline of the components that allowed us to integrate our use case with SODALITE covering already nine of the UML Use cases. In this second year, we further continued to develop the pipeline components to finish them and we integrated them with the previously released sub-pipeline.

The Snow use case focuses on the modeling activities showing that it is possible to define a

deployment model that fulfills the needs of the application (UC1), select the resources to be used at runtime (UC2), obtain generated IaC artefacts (UC3, UC4) and runtime images in the private image registry (UC16), and finally run and monitor the deployment (UC6-UC8) on the Cloud testbed. The training component of the skyline extractor component is deployed on the HPC testbed and optimized using MODAK (UC15). Additionally, partial redeployment (UC10) and UC9 Identify Refactoring Options (UC9) were executed by means of the WP5 toolset and platform resource discovery (UC17). Such mapping of the use case with the UML use cases is also presented in Table 16.

We modeled the pipeline using the knowledge base-empowered SODALITE IDE. The result of this is a JSON file, containing all the data and meta-data needed for the deployment and links to proper Ansible playbooks, that enable the setup and management of Snow components. This JSON is then sent to the iac-blueprint-builder component that converts it to a proper TOSCA blueprint that can be used by a TOSCA orchestrator (e.g., xOpera) to enact the deployment. At runtime, we verified that the generated blueprints could actually automate the deployment and configuration of the whole pipeline and we started monitoring the deployed pipeline**.**

Some components have response time requirements, in particular, we focused on the Skyline Extraction component by testing how it could be dynamically scaled in order to fulfill such time-constrained requirements. The current approach implemented in the Node Manager exploits heuristics and control-theory to dynamically change the CPU and GPU allocations according to the workload and context changes. The results retrieved with Skyline Extraction are promising and we plan to extend it to the other components of the pipeline that can be executed on heterogeneous resources (CPU/GPU). In particular, NodeManager was able to never violate the set SLA for Skyline Extraction during the run experiments, on the contrary the compared rule-based approach obtained 150 violations. In terms of resource consumption, NodeManager exploited 2515 core*second, 19% fewer than the rule-based approach. Overall, thanks to the smart load balancing and fine-grained resource allocation, the Node Manager was able to obtain fewer violations by using fewer resources with respect to the compared approach. Finally, the average response time of Skyline Extraction was equal to 0.217ms, 40% faster than the rule-based approach.

We also tested the reconfiguration by means of the deployment refactorer. For the use case we report 25% of refactoring scenarios are supported, which we plan to improve in the coming months by testing different deployment variants. In particular, the testing consisted of redeploying the application based on the resource usage of the VMs that host the application, to prevent over/under utilization of resources. The alert manager in the monitoring layer generates alerts indicating violations of SLAs or risks for potential violations and in response to these alerts, the Deployment Refactorer finds and enacts alternative deployment models, by leveraging deployment adaptation policies and resource (nodes) discovery capabilities. Thanks to the reconfiguration capability resource usage violations were prevented (e.g., if CPU usage reaches 80%, we move the application to a machine with higher CPU), the extent of success will be measured in further experiments.

The Skyline Extraction component uses a deep learning model that is the result of training a CNN we defined with a previously generated dataset. Modern environmental intelligence pipelines are based on different CNN models for some of its components. Training such a model is a time consuming and resource intensive task since it consists of testing several combinations of hyper-parameters and data augmentation techniques to obtain the best possible performance. For this reason, in the context of SODALITE, we aim to optimize the training process by means of the MODAK static optimization framework. In the next few months, we will use the Skyline Extraction model to benchmark the training process, by comparing the standard TensorFlow

container provided on DockerHub and the optimized one provided by MODAK. Initial benchmarking results in one of state-of-the-art architecture for image classification, ResNet-50[38], on the ImageNet[39] dataset, reported a 10% speedup measured based on the application run time as a metric and in coming months we will corroborate that such a speedup generalizes to other models, in particular to our Skyline Extraction component.

Currently, the main open issue is to optimize the connection of the components of the pipeline. The original and current deployment of Snow was limited to two VMs, and components were connected using local hard-drives and ad-hoc methods. We are going to address this issue by using more scalable and de-coupled ways to connect components (e.g., using queues or distributed DBs) so that the portability of the use case and its runtime management could be more effective. Furthermore, the dataset used for the training of the Skyline Extraction component requires data movement from source storage (e.g. on-site servers or cloud storage) into the HPC infrastructure, where the training is performed. This becomes challenging due to data transfer protocol incompatibilities, and therefore data management tools for such heterogeneity should be considered. In this context, a collaboration with RADON is pursued to test different strategies for the movement of data.

5.1.2 Clinical UC

The in-silico Clinical Trials use case reproduces real clinical trials in biomecanics by means of simulation to determine an optimal fixation and function of bone implant systems for patients with spinal conditions (e.g. disk displacement or prolapse). The use case is being developed by HLRS and was originally strictly HPC driven, i.e. it was realised as a workflow executed on a particular HPC infrastructure provider. This limits the adoption of the developed methodology in biomechanical clinical trials for medical device manufacturers or medical research institutes due to the specifics of the target execution environment of the use case.

Therefore, Clinical UC expects SODALITE to be beneficial in moving the process towards production-like environments with the following improvements:

- Increase the effectiveness and productivity of component deployment.
- Ease the adaptation to different IT-infrastructures (supercomputers, Clouds, on-premise) and different hardware.
- Lower the efforts for component integration.
- Lower the efforts for data management.

The rest of the subsection presents the SODALITE platform validation performed in the scope of Clinical UC during Y2 and concludes with updated SODALITE UML UC uptake for Clinical UC.

Clinical UC workflow orchestration: During the Y1, Clinical UC focused on orchestration aspects of SODALITE. SODALITE produced a set of TOSCA libraries[40] for execution of batch jobs on HPC clusters, managed by PBS Torque resource manager. Clinical UC validated these libraries by executing the workflow on the HLRS HPC testbed using the SODALITE orchestrator. In Y2, a more advanced orchestration scenario was validated. Clinical UC workflow was executed across multiple infrastructure targets: HLRS HPC testbed and EGI EC3[41] Torque/Slurm clusters. All targets were extended for experimentations with GridFTP, a data transfer protocol widely offered by HPC infrastructure providers, to enable data transfer between the targets. Therefore, SODALITE developed additional TOSCA libraries for data movement with GridFTP, thus enabling multitarget workflow execution of Clinical UC components with data dependencies.

Clinical UC benefited from this development as it allowed the distribution of the workflow execution from a single infrastructure target into multiple targets, utilising the capabilities offered by various providers. In this case, a less capable but more available virtual cluster (EGI EC3) was used for UC components (Density Mapping and Boundary Condition) that do not demand a lot of

resources (short jobs). Bare-metal resources (HLRS HPC testbed) with installed MPI libraries for parallelisation was used for the more compute-demanding MPI-parallel component (Probabilistic Mapping). The setup and workflow execution steps are presented in Figure 20 and the demonstration can be found here: https://www.youtube.com/watch?v=vAIt8-t4hhM.

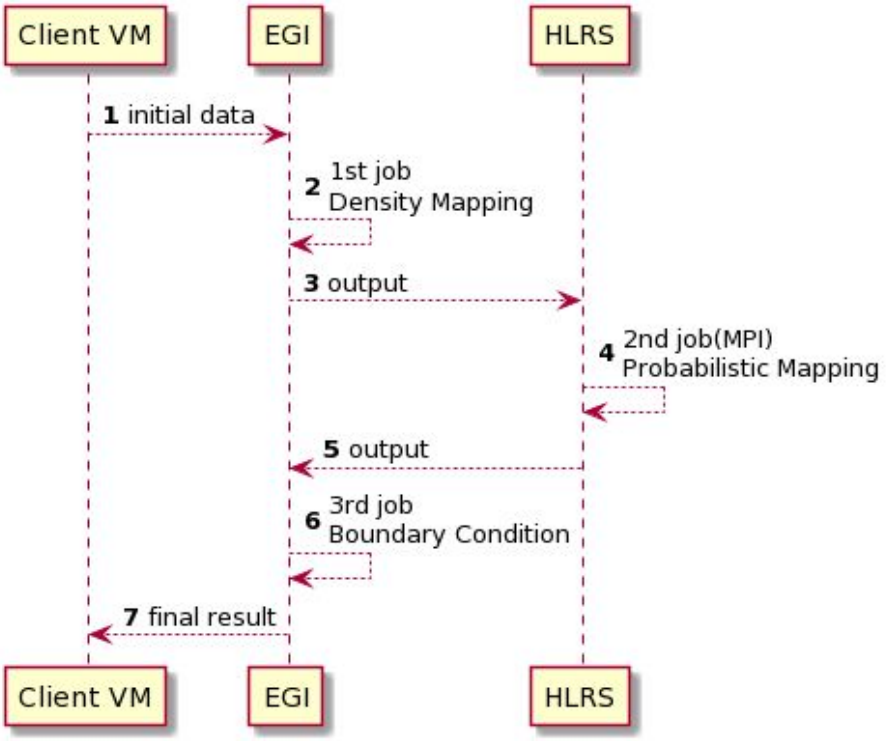

Figure 20 - Clinical UC workflow execution on HLRS HPC testbed and EGI EC3 cluster with data transfers using GridFTP

As it can be observed, the experiments targeted only clusters with resource managers such as Torque and Slurm, only partially satisfying workflow orchestration requirements. Although indeed the original workflow has been improved with distributed execution on different HPC clusters, the improvements towards workflow orchestration over the targets other than HPC (such as Cloud or remote servers) are yet to be investigated in Y3, when all components of Clinical UC are developed and integrated into the final workflow. In particular, a more generalized workflow will be validated.

Same applies to the data management support, which is limited to GridFTP. In production, there can be cases where data should be acquired from remote repositories, where GridFTP usage is not common, e.g. clinics or medical research facilities. Therefore, data management should also be generalized and will be addressed in the context of RADON collaboration in Y3.

Clinical UC workflow optimization: Together with optimization experts (Quality Experts), we focused on optimization of the Code_Aster Solver component, which is contributing the most to the total execution of the workflow. The single thread execution in the optimized container was improved by 3% compared to the official solver container as reported in Section 5.3.1. Furthermore, with the build of parallel Code_Aster, additional optimised container images are being prepared for parallel execution of the Solver, which is proven to reduce the execution time significantly, as shown in Section 4.2.3.

Clinical UC workflow modelling: SODALITE IDE offers a context-assistance and models validation that was useful during the modelling of the Clinical UC deployment. Moreover, SODALITE IDE offers optimisation specification, which allows to create optimisation recipes (with the help of Quality

Experts) and apply optimisations to a particular application component, which in turn is executed using optimised container runtime.

As a validation scenario, using SODALITE IDE, we modelled and deployed a single target use case workflow execution on HLRS HPC testbed with the optimisation applied to the Probabilistic Mapper for parallel execution. Resource Experts have provided HPC resource models and models for workflow execution. Clinical UC developers acted as Application Ops Experts and created an AADM (Abstract Application Deployment Model) for the use case deployment, utilising provided resource models. Quality Experts helped to develop optimisation recipes to enable MPI parallelisation for Probabilistic Mapper, during runtime of which, it was built and executed within optimised container runtime.

During the development of AADM for Clinical UC, SODALITE IDE assisted with modelling, e.g. offering available node types and resolving requirements for node templates. Inconsistencies in deployment models (e.g. mismatch of node types in requirements) were also checked and reported back instantly at the development time. Since initially we started deployment modelling using TOSCA with a simple YAML editor, we found the usage of the SODALITE IDE extremely convenient, error reducing and saves effort for failure resolution and component integration. As the next step, final validation of SODALITE IDE will be performed in Y3 by execution of complete workflow with data movement across multiple various targets.

Overall during Y2, Clinical UC extended the uptake of SODALITE UML UCs, concerning modelling and optimisation aspects, shown in Table 16. As such, we defined the Clinical UC deployment model in the SODALITE IDE (UC1) and selected required resources for the use case execution (UC2). Further, we statically optimised use case components (Solver and Probabilistic Mapper) (UC12, UC15). Based on the defined deployment model, the deployment IaC is generated (UC3) and validated (UC4), and the use case is executed (UC6, UC7). Monitoring of jobs execution via HPC exporter (UC8) was validated, however, further integration should be investigated.

5.1.3 Vehicle IoT UC

The Vehicle IoT use case involves the development and provisioning of services for connected vehicles, running both in the Cloud as well as directly at the Edge (via a dedicated Edge Gateway physically installed into participating vehicles). Each Edge Gateway contains different hardware configurations, including different heterogeneous accelerators that can be leveraged by services. A key challenge for the use case is that the application developer of a given service has little to no insight into the precise hardware configuration that exists in the vehicle, and so these heterogeneity challenges must be handled directly (and transparently) by SODALITE.

Each Edge Gateway participates as an Edge node in a Kubernetes cluster and can have different applications deployed at any given time. This creates a number of unique challenges, in that:

- Not all applications are created equal: Some applications may have a hard dependency on a particular accelerator type, while others (particularly those involving Edge-based AI/ML inference models) may deploy their models on a range of different accelerators, depending on availability.
- As load increases, the increase in ambient temperature within the Gateway housing can bring accelerators outside of their safe operating temperature limits, resulting in inference failure and other hard to debug problems that are not immediately apparent to the application developer. It may, therefore, be necessary to throttle the deployment, redeploy the application onto a more thermally efficient (or tolerant) accelerator, migrate the application from the Edge to the Cloud, or temporarily suspend execution of the application in order to avoid a service lapse / SLA violation.

Thermal-based Kubernetes deployment refactoring: An application making use of the EdgeTPU accelerator for running its inference model was deployed into a vehicle Edge node as a Kubernetes deployment, and system load was generated in order to drive up the internal ambient temperature within the Gateway housing. The EdgeTPU itself, notably, begins to generate inference failures and other erratic behaviour when its internal temperature exceeds 85°C, and physical damage to the EdgeTPU package itself is possible via thermally-induced microfractures when the package temperature begins to exceed 95°C (in this case, an internal thermal trip point may simply gate off the internal voltage supply in order to prevent hardware damage). A pre-configured alerting rule has been defined in Alertmanager, which informs the SODALITE refactorer of the urgent need to refactorer the deployment. The SODALITE image builder has been leveraged to generate different application container variants with different performance profiles, which the refactorer is able to switch between. The application itself, originally deployed with the maximum performance profile, is redeployed with a reduced performance version, subsequently reducing the internal device temperature. This is exemplified in Figure 21 below:

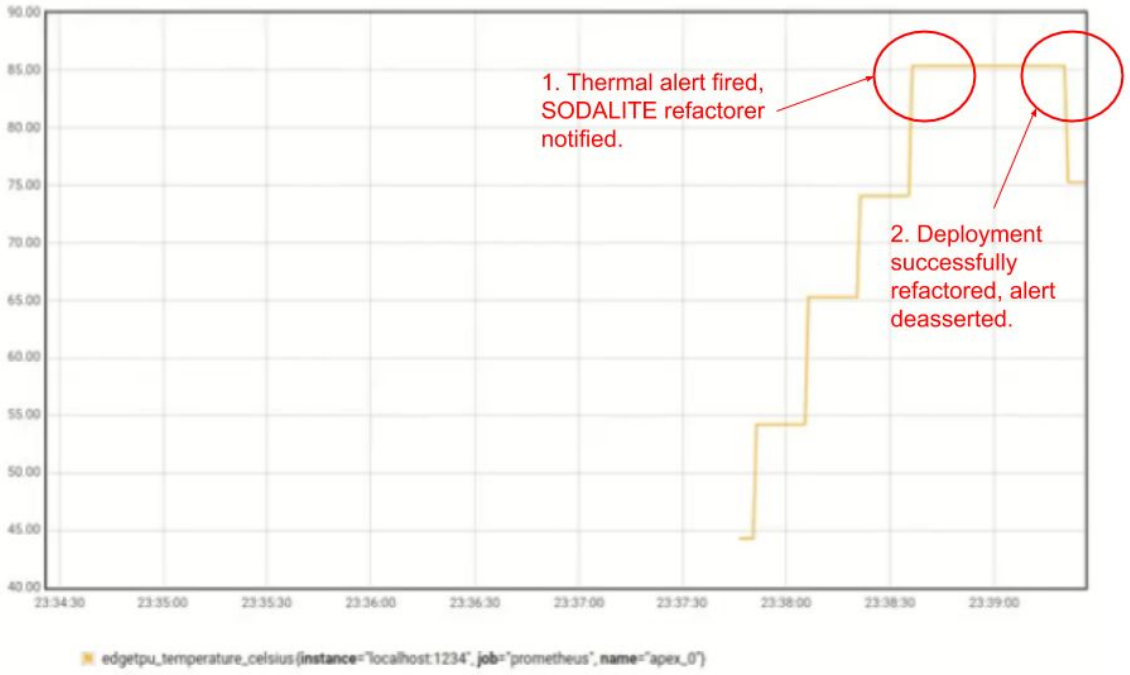

Figure 21 - Kubernetes thermal-based deployment refactoring with SODALITE

While the normal operating temperature of the EdgeTPU has been observed to be between 45-65°C, heavy load generated by other applications deployed in the Edge Gateway has been found to easily trigger the 85°C trip point, meaning that this is a scenario that will have to be accounted for in any operational environment. Presently the refactorer is only triggered based on predefined alerting rules for the given accelerator type and refactors the deployment by switching between different application container variants that configure different internal device clock rates. In Y3, it is expected to use additional metrics in order to build a contextual understanding of the root cause (e.g. is the source of the spike in ambient temperature originating from the CPU or GPU thermal zones rather than from EdgeTPU load?), and to apply ML techniques to contextually optimize deployment blueprints, thereby increasing the level of sophistication in possible refactoring options and mitigation measures. A full demonstration and walk-through of the steps taken are presented in a YouTube video: https://www.youtube.com/watch?v=dotlBOn1jmI

At this stage, the Vehicle IoT use case can be seen to make effective use of the SODALITE run-time monitor, refactorer, and image builder. While work done in the development of Edge-based Kubernetes controllers and labellers has provided a basis for resource discovery, more work must

be done to integrate this directly with SODALITE. Initial experiments with the SODALITE IDE have demonstrated that it is possible to develop and deploy simple applications into the Edge environment, but additional work on extending the blueprint definition to support the Edge characteristics is still required. This work must be carried out before the remainder of the SODALITE technology stack can be reasonably applied to the use case's Kubernetes-based operational environment in Y3.

5.2 Platform Evaluation: Modelling Abstractions

5.2.1. Abstraction of application and infrastructure (KPI 1.1)

This evaluation refers to the capability of the modelling layer to support the defined use cases in terms of abstract application and infrastructure structures

5.2.1.1 Analysis

According to the evaluation plan [D2.2], querying the SODALITE Knowledge base and the existing AADMs and RMs in the SODALITE IDE repository[42], the following metrics has been computed:

1. # RMs: number of created resource models, containing type definitions for reusable infrastructure resources
2. # AADMs: number of created abstract application deployment models, containing component (template) definitions
3. # type definitions: total number of existing type definitions for infrastructure resources
4. # components (templates): total number of existing component (template) definitions

5.2.1.2 Results

Computed metrics at M24 are shown in the table below. Required use case definitions for infrastructure resources and application components have been created, excepting those definitions required for Vehicle IoT Kubernetes resources and associated components.

5.2.1.3 Achieved KPIs

At M24, over 66% of required specifications (i.e. abstractions) for infrastructure resources and application components have been modelled, all those required by Snow and Clinical use cases, and a number of them (not all as Kubernetes related ones are pending) for Vehicle IoT.

KPI 1.1	Abstraction of application and infrastructure.
Target	Lower bound is 25% coverage of all application and infrastructure scenarios in the scope of SODALITE case-studies.
Deadline	M24
Metric	**Value M24**
# resource models	28
# AADMs	14
# type definitions	172
# components (templates)	148

Total	364 (66% of the required ones)

5.2.2. Abstraction of Infrastructure Performance Patterns (KPI 1.2)

To measure the achievement of KPI 1.2, we have analysed the demonstrating use cases and identified the required performance patterns as they have been stated at the beginning of the project.

5.2.2.1 Analysis

We use the use case requirements to map this and calculate the lower bound. There is not an automated process. The collection is performed manually by inspecting the requirements formulated by case study users.

5.2.2.2 Results

The results are reported for each use case:

- Snow use case
  - Increase thought of the images analysis with fast I/O storage
  - Increase machine learning training and interference performance by exploiting the use of GPU computing
- Clinical use case
  - Reducing the amount of processing data after the material mapping
  - Support execution of MPI parallel programs
  - Support fast networking performance
  - Support fast I/O performance

The Vehicle IoT use case has not been analysed yet.

5.2.2.3 Achieved KPIs

The abstraction of the infrastructure performance patterns contributes to KPI 1.2, which is due on M33. For M24, the values are:

KPI 1.2	Abstraction of Infrastructure Performance Patterns.
Target	Lower bound is 80% of all performance patterns found in HPC and Cloud infrastructures.
Deadline	M33
Metric	**Value M24**
% of use case performance requirements modeled in WP3	80% for Clinical UC
% of use case performance requirements modeled in WP3	92% for Snow UC
% of use case performance requirements modeled in WP3	to be measured for Vehicle IoT UC

5.2.3. Abstraction of execution constraints and possibilities (KPI 1.3)

This evaluation refers to the execution requirements and constraints on compute, memory, network, storage, etc, that have been modeled in the context of the demonstrating use cases.

5.2.3.1 Analysis

Each use case is related with specific execution requirements and constraints. For checking the fulfillment of this KPI, the used metric is the percentage of execution constraints and possibilities that have been modelled in the Modelling Layer. The results have been collected manually by inspecting the requirements set by the use case owners.

5.2.3.2 Results

The execution requirements and constraints for all the use cases are shown in the tables below. The execution constraints that have not been abstracted for M24 have been marked as red and their absence, along with a timeplan, provided in the specific use case section above.

Snow use case

Execution Requirements and Constraint	**Description**
Parallel workflow	HPC workflows to be executed in parallel
Storage	Storage such as ssd disk
GPU	GPU capability
Response/Real time	The process to be done in real time
Service	Execution of services by exposing ports, modelling dockers
Availability	A resource is re-allocated based on a policy.

Clinical use case

Execution Requirements and Constraint	**Description**
Parallel workflow	HPC workflows to be executed in parallel
Memory -RAM	Memory limit to be modelled
MPI	Support execution of parallel MPI programs
Serial	Not parallel workflows
Network	1) Fast Networking Performance 2) Transmission of large messages e.g.

	through routers, switches
Fast Storage	Support fast I/O performance

Vehicle IoT use case

Execution Requirements and Constraint	**Description**
Heterogeneous Accelerators	Support execution on GPU, EdgeTPU, NCS2
Edge	Edge Computing

Note that in the Vehicle IoT use case, a range of execution possibilities exist, particularly with regards to the different types of accelerators. These must still be modelled going into Y3, but are not expected to present any specific challenges, as the additional accelerator types can be derived from the GPU definition, which is already in use by the other use cases. The Edge-based deployment constraint requires the integration of Kubernetes support, which will only begin at the beginning of Y3. Limited experimentation has, however, been carried out with the AADM model for deployment into Kubernetes clusters, and deployment from the Cloud-based orchestrator to the Edge-based Kubernetes cluster has also been validated. It is not expected that there will be any technical barriers to enabling this from the IDE outside of ensuring the relevant components have been updated and integrated.

5.2.3.3 Achieved KPIs

The abstraction of the execution constraints and possibilities contributes to KPI 1.3, which is due on M33. For M24, the values are:

KPI 1.3	Abstraction of execution constraints and possibilities.
Target	Lower bound is coverage of 80% of execution scenarios.
Deadline	M33
Metric	**Value M24**
% execution possibilities modelled in WP3	83% for Snow UC
% execution possibilities modelled in WP3	66% for Clinical UC
% execution possibilities modelled in WP3	to be measured for Vehicle IoT UC

5.3 Platform Evaluation: Performance

5.3.1.Static optimisation (KPI 2.1)

Static optimization is the main objective of MODAK and is the mechanism that contributes to KPI 2.1 (increase of abstracted application performance on abstracted infrastructure by using Infrastructure performance abstraction patterns). In this section, we present the performance benefit introduced by the MODAK static optimization framework on two types of applications: an AI and an HPC example.

For the AI example, we evaluate MODAK using image classification training and time the execution of a set number of epochs. We chose to train on the ResNet-50[43] residual network for GPU workloads. Optimisations were performed by building AI frameworks from source, using graph compilers, as well as building the same graph compilers from source.

AI frameworks use intermediate representations (IR) to represent the neural network models as computational graphs, with nodes representing tensor operations and edges the data dependencies between them. Graph compilers optimise this computational graph and then generate an optimised code for a target hardware/backend, thus accelerating the training and deployment of deep learning models. XLA (Accelerated Linear Algebra)[44] is a TensorFlow specific graph compiler that accelerates linear algebra.

For the HPC applications evaluation, we used Code_Aster[45], a standalone thermo-mechanical solver used in Clinical UC. The Code_Aster (v14.4.0) uses finite element methods to compute a solution that shows the strain and stress distribution within the simulated structures, as well as the displacement field for the simulation of two human vertebrae. Code_Aster has multiple dependencies including hdf5, mumps, numpy, OpenBLAS, metis, ptscotch, petsc, and parmetis, and containers provide an efficient way to deploy.

5.3.1.1 Experiment description and setup

We performed the benchmarking of the AI applications on a SODALITE HPC testbed set up at HLRS, the research and supercomputing center affiliated to the University of Stuttgart. The testbed consists of a front-end node running Torque, and five compute nodes, each hosting an Nvidia GeForce GTX 1080 Ti GPU, an Intel(R) Xeon(R) CPU E5-2630 v4 processor, and 125GB of main memory.

HPC application results were obtained on the MC partition of the Cray XC50 "Piz Daint" supercomputer at the Swiss National Supercomputing Centre (CSCS). Each node of the system is equipped with a dual-socket CPU Intel Xeon E5-2695 v4 @ 2.10GHz (18 cores/socket, 64/128 GB RAM).

We used Singularity v3.5.3. Each test was run 3 times and the average of the runtimes calculated. Runtime fluctuations were found to be below 2%.

5.3.1.2 Results

Most AI frameworks support and provide Docker containers for different targets and can be readily used for deployment. Due to Docker uptake this is our baseline method of deploying AI training applications. For AI frameworks we compare the performance of MODAK optimised containers built under Singularity with that of the official container available on DockerHub. We built TensorFlow for the MODAK containers with XLA graph compiler enabled. We used the latest standard release version 2.1. The network was trained on the ImageNet database, which consists of more than 14 million hand-annotated images in 20,000 categories. Then, we used single

precision, a batch size of 96, and 3 epochs to train. The MODAK container shows 10% improvement in performance for the execution time.

For Code_Aster, we compared the performance of the MODAK optimised container with that of the official Code_Aster container, available from https://github.com/codeaster/container. On a single thread, the MODAK containerised application had an average execution time of 725 seconds, while the official Code_Aster container executed for 745 seconds. This is a 3% speed-up.

5.3.1.3 Achieved KPIs

The static optimisation contributes to KPI 2.1, which is due on M30. For M24, the values are:

KPI 2.1	Increase of abstracted application performance on abstracted infrastructure by using Infrastructure performance abstraction patterns
Target	Application performance increased by 15%. The performance metric to be used will depend on the specific case study.
Deadline	M30
Metric	**Value M24**
Speedup measured based on application run time as a metric.	**10%** for ResNet-50
Speedup measured based on application run time as a metric.	**3%** for Code-Aster Solver

5.3.2 Reconfiguration: Deployment Refactorer (KPI 2.2)

Deployment Refactorer enables reducing SLA violations and improving resource usage and application performance by adapting the deployment model of the application at runtime. The monitoring layer generates alerts and events, indicating potential SLA violations. In response to these events, the Deployment Refactorer decides and makes necessary changes to the current deployment, for example, finding resources (nodes) that offer better security or have more resources, and use those resources.

Based on the above considerations, the Deployment Refactorer contributes to KPI 2.2, increase of concretized (deployed) application performance running on targeted infrastructure through Predictive Deployment Refactoring.

5.3.2.1 Experiment description and setup

The first set of experiments focus on deployment refactoring/adaptation requirements of Vehicle IoT and Snow use cases. For Vehicle IoT use case, two different deployment adaptation scenarios were implemented. In the first scenario, the application is redeployed in response to changes in legal jurisdiction to maintain both service continuity and meet the compliance requirements as vehicles travel between countries. The second demonstrated the capability of the Edge-based monitoring and alerting to throttle an application deployment that has exceeded thermal tolerances. By switching between different variants of the inference application containers at different thermal trip points, the risks of rising temperature inducing inference failure is prevented.

In the context of Snow use case, to prevent over/under utilization of resources, the application is redeployed based on the resource usage of the VMs that host the application. In each of these experiments, the alert manager in the monitoring layer generates alerts indicating violations of SLAs or risks for potential violations, for example, *HighCPULoad*, *LocationChanged*, and *HighEdgeTPUDeviceTemperature*. In response to these alerts, the Deployment Refactorer finds and enacts alternative deployment models, by leveraging deployment adaptation policies and resource (nodes) discovery capabilities.

The second set of experiments focus on evaluating the accuracy and efficiency of our machine learning based approach to predicting the performance of an application with many different deployment alternatives/variants. Due to cost and time constraints, it is usually infeasible to measure the performance of each possible deployment variant. Thus, we provide an approach for modeling and predicting the performance of all the valid deployment variants of an application based on the observed performance of a minimal subset of the variants. We validated our approach with the popular RuBiS benchmark application and Google Cloud Platform. The application had 93 deployment variants (various combinations of 16 deployment options for individual components of the application). The performance data were collected through benchmarking. D5.2 provides more information on our performance modeling approach and its validation.

5.3.2.2 Results

Figure 22 shows the accuracy of the performance models and the impact of the sample size (the subset of deployment variants used for measuring performance data). We used three different machine learning algorithms, and all three performed well even for the initial sample.

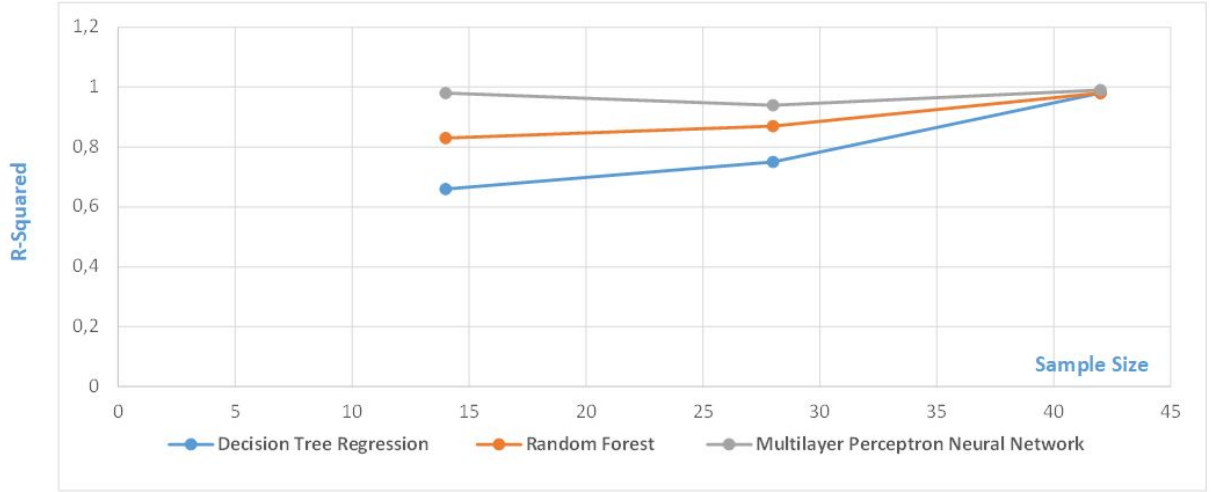

Figure 22 - Impact of sampling on Performance of Performance Prediction Models

5.3.2.3 Achieved KPIs

The Deployment Refactorer contributes to KPI 2.2, which is due on M30. For M24, Refactorer's contributions to the KPI 2.2 are:

KPI 2.2	Increase of concretized (deployed) application performance running on targeted infrastructure through Predictive Deployment Refactoring.
Target	Application performance increased by 15%. The performance metric to be used will depend on the specific case study.

Deadline	M30
Metric	**Value M24**
% of Refactoring Scenarios Supported	[Vehicle UC] **75%** (preventing violation of placement constraints, under/over usage of resources, and device inference failures) [Snow UC] **25%** (preventing under/over usage of resources)
Accuracy and Efficiency of Predicting Performance of Deployment Alternatives	**98%, 96% and 99%** respectively for Three Different Performance Prediction (ML) Models Performance measurements for **10%-35%** of all the deployment variants is sufficient for building a good performance prediction model

5.3.3 Reconfiguration: runtime SLA violation (KPI 2.2)

While the deployment refactorer addresses the long term problem of finding a more effective deployment configuration, the Node Manager is in charge of improving performance on the short term by continuously controlling the execution of applications and applying continuous corrections. As such, it also contributes, together with the Deployment Refactorer, to KPI 2.2, increase of concretized (deployed) application performance running on targeted infrastructure through Predictive Deployment Refactoring.

5.3.3.1 Experiment description and setup

A prototype of Node Manager was implemented and evaluated using four benchmark applications: Skyline Extraction from Snow UC, ResNet, GoogLeNet, and VGG16. To run the experiments, NodeManager was deployed on a cluster of three virtual machines on Microsoft Azure: one VM of type HB60rs with a CPU with 60 cores and 240GB of memory, and two VMs of type NV 6 equipped with a NVIDIA Tesla M60 GPU and a CPU with 6 cores and 56GB of memory. An additional instance of type HB60rs was used for generating the client workload. Different shaped, highly varying synthetic workloads were tested in all the experiments run and the different applications were run in random combinations concurrently on the servers.

5.3.3.2 Results

The first type of experiment conducted for the NodeManager was about varying either the input workload or the set-point of the system to test the ability of Node Manager to rapidly adapt the resource allocation to the new state. Results showed that the NodeManager is able to efficiently adapt to different unforeseen conditions. In Figure 23, as an example, is shown how Node Manager quickly reconfigures the resources when the SLA is changed at runtime.

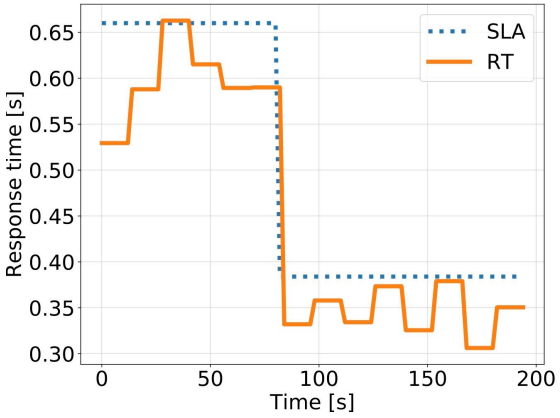

Figure 23 - Node Manager reacts to a change of SLA at runtime

Node Manager was compared with a rule-based approach that schedules incoming requests using a round-robin approach on available CPUs and GPUs and dynamically scales the resources using a rule-base engine. Different synthetic workloads were tested and the Node Manager outperformed the baseline in all the experiments obtaining overall 96% fewer SLA violations while using 15% fewer resources. Figure 24 shows the different behavior of the systems (left rule-based, right Node Manager) with the same workload when all the four applications were running concurrently. While Node Manager can quickly react to changes, rule-based approach often violates the SLAs for applications ResNet and VGG16.

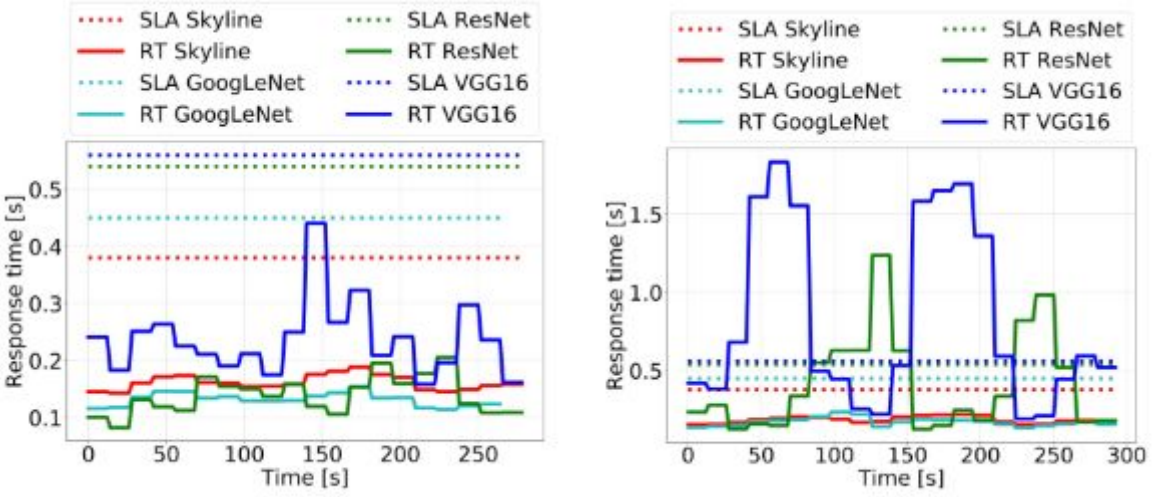

Figure 24 - Comparison between Node Manager (left) and Rule-based approach (right)

By adopting the Node Manager, use case owners can deploy components that exploit heterogeneous resources, set constraints on their response times and have the platform automatically managed for optimizing resource allocation and fulfill the desired goal. Node Manager is able to control multiple applications at the same time and to govern potential resource contention scenarios among concurrent applications. As clearly shown in Figure 24, Node Manager outperforms heuristic-based control by order of magnitude. The SLA violations are minimized (96% improvement) and the resources are precisely allocated to the different containers (15% improvement).

5.3.3.3 Achieved KPIs

The Node Manager contributes to KPI 2.2, which is due on M30. Node Manager uses two baselines for evaluating its performance.

Kubernetes is used as a baseline to compare the scaling capabilities of containerized systems. In particular, NodeManager is compared against Kubernetes Horizontal Pod Autoscaler and Vertical Pod Autoscaler. Given the lack of support for GPU autoscaling of Kubernetes, to evaluate the efficiency of the management of heterogeneous resources, NodeManager was compared with a rule-based approach.

KPI 2.2	Increase of concretized (deployed) application performance running on targeted infrastructure through Predictive Deployment Refactoring.
Target	Application performance increased by 15%. The performance metric to be used will depend on the specific case study.
Deadline	M30
Metric	**Value M24**
Scaling capabilities	*Node Manager*: combined CPU and GPU allocation based on control-theory and lightweight heuristics for containerized application. Fast vertical scaling of resources (nondisruptive re-configuration at each second) *Baseline*: no support for GPU allocation, vertical scalability needs disruptive action (i.e., container restart), slower updates **Node Manager provides combined CPU and GPU management**
SLA Violations	*Node Manager*: 10 *Baseline*: 270 **~96% fewer violations**
Allocated resources (cores * seconds)	*Node Manager*: 4148 *Baseline*: 4829 **~15% fewer resources used**

5.4 Platform Evaluation: Usability (KPI 3.1 and KPI 3.2)

To assess the usability of SODALITE, we focused on the IDE that, being the modeling environment, is the one that is closest to our end users (AOEs). We performed some controlled experiments with three different types of users, namely non-expert users, TOSCA experts, and SODALITE use case owners, with the objective to receive feedback from multiple viewpoints. We collected feedback on perceived ease of use, perceived usefulness and intention to use, which are the common factors deciding the user acceptance of a particular technology[46].

The goal of these experiments is to assess the usability of SODALITE in terms of IaC development and management cost and effort (KPI 3.1 and KP 3.2). Non-expert users and TOSCA experts focused on the development of IaC for a machine learning application, which consists of 1) a database that stores training data, 2) a component that trains a machine learning model 3) a

repository that stores trained machine learning models, 4) a component that makes predictions/inferences based on the trained models. The use case owners used their corresponding use cases.

5.4.1 Normal users (inexperienced in TOSCA)

The goal of this experiment is to compare the time needed to define correct deployment code without and with SODALITE. We performed the experiment with the students from a master-level course.

5.4.1.1 Experiment description and setup

Each experiment participant went through the following steps:

1. Training on the usage of the TOSCA language and on the usage of the SODALITE IDE. This training occurred before the actual experiment, exploiting tutorial videos we have prepared for this purpose.
2. Development of the following two exercises.
    - Exercise A - Development of a deployment blueprint in TOSCA for the machine learning application using a plain YAML editor;
    - Exercise B - Development of an abstract application deployment model (AADM) using the SODALITE IDE for the same machine learning application.

    Half of the group realized the two exercises in the order first A and then B, half realized them in the opposite order, first B and then A.
3. An anonymous questionnaire to get the feedback from the participants on perceived ease of use, perceived usefulness, and intention to use.

5.4.1.2 Results

There were 9 participants. All participants were able to complete the tasks successfully.

Perceived Ease of Use. The participants considered the user interface of SODALITE IDE very intuitive and easy to follow, and context-aware content assistance support was very helpful in developing deployment models correctly and quickly. However, they also noted that the visualization of the deployment topology needs further improvements with regards to complexity and usability. The following table shows the distribution of answers comparing the perceived ease of use of the SODALITE IDE against the writing of the TOSCA blueprint with a YAML Editor.

Perceived Ease of Use	**SODALITE IDE**	**TOSCA with a YAML Editor**
Very complex and difficult to use	0	0
Difficult to use	1	3
Neither difficult nor easy to use	2	3
Easy to understand	4	3
Very clear and easy to understand and use	2	0

Perceived Usefulness. The participants considered the SODALITE IDE useful, for example, reducing effort, ease of understanding even complex deployment models, and effective in defining the deployment code correctly and intuitively. The following table shows the distribution of answers comparing the usefulness of the SODALITE IDE against writing of the TOSCA blueprint with a YAML Editor.

Perceived Usefulness	**SODALITE IDE**	**TOSCA with a YAML Editor**
Not Useful at all	0	0
Not Useful	0	0
Neither not useful nor useful	0	4
Useful	4	5
Very useful	5	0

Intention to Use. The participants also reported their opinions on their intention to use the SODALITE IDE or a YAML editor in the future when they will be working at a company developing deployment code/models. Figure 25 and Figure 26 show the results of the responses from the participants for the SODALITE IDE and YAML editor, respectively:

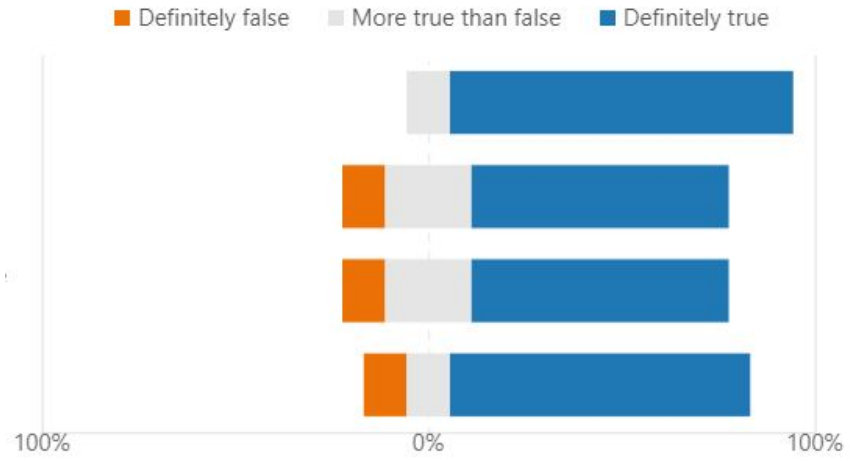

Figure 25 - Intention to use the SODALITE IDE for defining deployment models: 1) A potential in the adoption of the SODALITE approach, 2) Recommending the SODALITE IDE for others, 3) Using the SODALITE IDE at a company, 4) Becoming skilled in using the SODALITE IDE

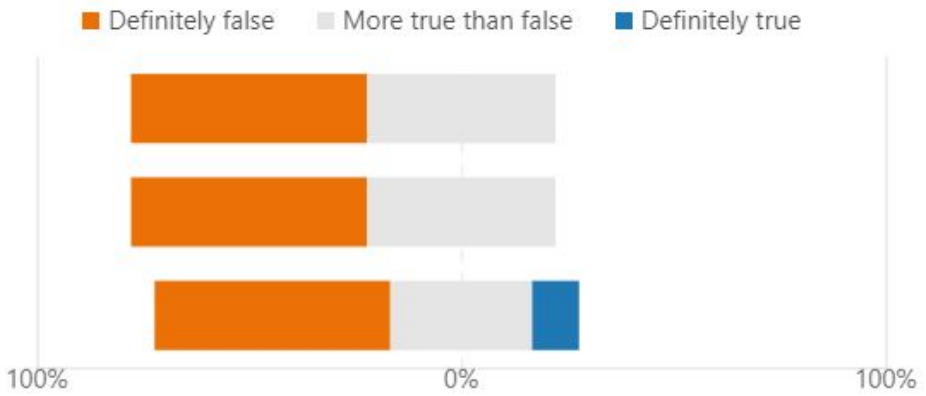

Figure 26 - Intention to use the YAML Editor for defining deployment models: 1) Continually using the YAML editor instead of the SODALITE IDE, 2) Using the YAML editor at a company, 3) Becoming skilled in using the YAML Editor

5.4.2 TOSCA experts

5.4.2.1 Experiment description and setup

The experts first developed the AADM (the deployment model) for the machine learning application using SODALITE IDE. Next, they modified the AADM by adding new nodes and relationships, and updating and removing some existing nodes and relationships. Then, they did the same activities using plain TOSCA and a YAML editor. Finally, they reported their experience and feedback using an anonymous questionnaire.

5.4.2.2 Results

There were 5 participants. The average level of TOSCA proficiency was 3.8 (from 5). All participants were able to complete the tasks successfully.

Activity	AADM (with SODALITE IDE)	TOSCA (with YAML Editor)
Development	32.14 mins	44.47 mins
Modification	7.67 mins	9.5 mins

Perceived Ease of Use. The TOSCA experts considered both the SODALITE IDE and a YAML editor as relatively easy to use. They also noted the current support for the content assistance, auto completion and complex data types in the IDE as well the documentation of the IDE should be improved.

Perceived Ease of Use	SODALITE IDE	YAML Editor
Very complex and difficult to use	0	1
Difficult to use	0	0
Neither difficult nor easy to use	3	2
Easy to understand	2	1

Very clear and easy to understand and use	0	1

Perceived Usefulness. The participants generally considered the SODALITE IDE is useful (compared with a YAML editor) for defining deployment models due to the support for partial auto completion, validation, and content assistance.

Perceived Usefulness	**SODALITE IDE**	**YAML Editor**
Not Useful at all	0	0
Not Useful	0	2
Neither not useful nor useful	0	1
Useful	4	0
Very useful	1	2

Intention to Use. The participants also reported their opinions on their intention to use the SODALITE IDE or a YAML editor in the future when they are working at a company developing deployment code/models. Figure 27 and Figure 28 show the results of the responses from the participants for the SODALITE IDE and YAML editor, respectively:

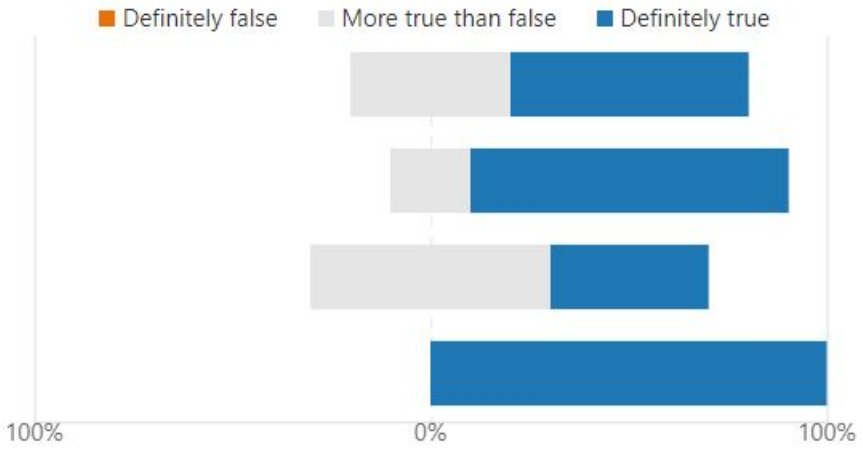

Figure 27 - Intention to use the SODALITE IDE for defining deployment models: 1) A potential in the adoption of the SODALITE approach, 2) Recommending the SODALITE IDE for others, 3) Using the SODALITE IDE at a company, 4) Becoming skilled in using the SODALITE IDE

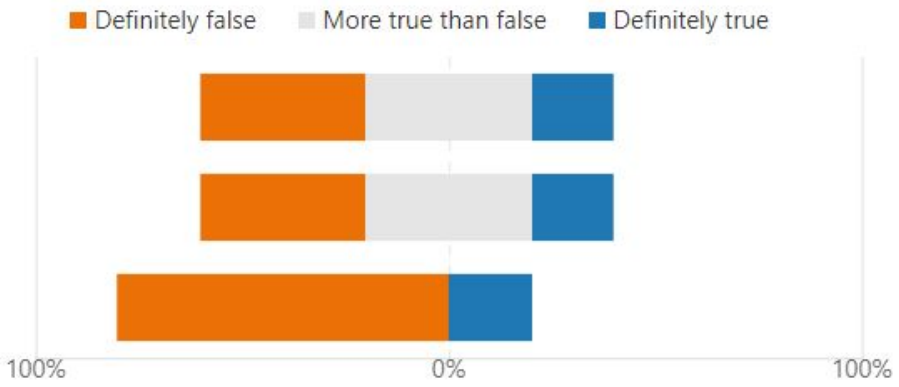

Figure 28 - Intention to use the YAML Editor for defining deployment models: 1) Continually using the YAML editor instead of the SODALITE IDE, 2) Using the YAML editor at a company, 3) Becoming skilled in using the YAML Editor

5.4.3 Use case owners

5.4.3.1 Experiment description and setup

The use case owners have received the M18 version of the AADMs for their use cases and asked to study these AADMs. Next, they were asked to modify the AADMs by adding new nodes and relationships, and updating and removing some existing nodes and relationships. Finally, they reported their experience and feedback using an anonymous questionnaire.

5.4.3.2 Results

There were 4 participants. All the participants were able to navigate through the AADMs associated with their case studies without specific problems. Three participants were able to extend and modify the AADM without significant effort, while one participant was only partially successful in changing the AADM.

Activity	**Time (Number of Participants)**
Development	15 mins (2) Between 15 mins and 30 mins (2)
Modification	15 mins (1) Between 30 mins and 1 hour (3)

Perceived Ease of Use. The participants considered that it is easy to use the IDE although they found that more assistance through documentation and tooltips with suggestions would make using the IDE easier, particularly for users with no prior experience with Eclipse.

Perceived Ease of Use	**SODALITE IDE**
Very complex and difficult to use	0
Difficult to use	0

Neither difficult nor easy to use	1
Easy to understand	3
Very clear and easy to understand and use	0

Perceived Usefulness. The participants considered the SODALITE IDE is useful as it enables defining the whole deployment model and components relationship with one tool. The ability to show errors and warnings in the code also made the IDE more useful.

Perceived Usefulness	**SODALITE IDE**
Not Useful at all	0
Not Useful	0
Neither not useful nor useful	1
Useful	2
Very useful	1

Intention to Use. The participants also reported their opinions on their intention to use the SODALITE IDE or a YAML editor in the future when they are working at a company developing deployment code/models. Figure 29 shows the results of the responses from the participants:

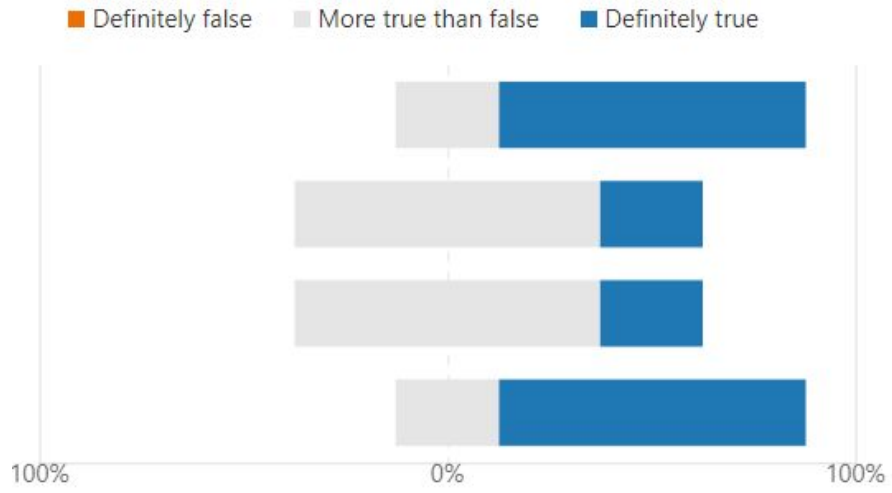

Figure 29 - Intention to use the SODALITE IDE for defining deployment models: 1) A potential in the adoption of the SODALITE approach, 2) Recommending the SODALITE IDE for others, 3) Using the SODALITE IDE at a company, 4) Becoming skilled in using the SODALITE IDE

5.4.4 Achieved KPIs

The three experiments presented in the previous sections have contributed to the assessment of both KPI 3.1 and KPI 3.2 as described in the following tables.

As regards to KPI 3.1, the experiments with TOSCA experts showed the SODALITE can help to achieve 27.73% improvement over the baseline. The experiments with each external group showed

that the users consider the SODALITE IDE is very useful, easy to use, and has a high potential for its adoption. As regards to KPI 3.2, the improvement was 19.26% over the baseline, which was approximately 10% below the target for the KPI. Based on the feedback from the users, some missing features of the IDE (some of them are planned for M30) and the partial stability and incomplete documentation of the IDE have negatively impacted on the user's effort and time. This may indicate that KPI 3.1 and KPI 3.2 need to be further evaluated once the SODALITE IDE is feature-complete, and officially released.

KPI 3.1	Reduction in software and/or application development time and cost.
Target	Lower bound target is 10% improvement over the baseline and will be evaluated through external parties where possible. The improvement will be measured by considering the time needed to develop an application manually and then with SODALITE.
Deadline	M24
Metric	**Value M24**
Time needed to develop a the complete blueprint	TOSCA Experts: ● SODALITE IDE: 32.14 mins (average) ● YAML Editor: 44.47 mins (average) **27.73%** Improvement with the SODALITE IDE TOSCA experts developed AADMs and TOSCA models of an application. Use Case Owners: ● 18.75 mins (average) Using SODALITE IDE, the user case owners explored the deployment model for the M18 version of their case study.
Perceived ease of use	1 to 5 scale where 5 - very clear and easy to understand and use; 1- very complex and difficult to use. Normal users ● SODALITE IDE : 4.22 ● YAML Editor: 3 TOSCA experts ● SODALITE IDE : 3.4 ● YAML Editor: 3.2 Use case owners ● 3.75
Perceived usefulness	1 to 5 scale where 5 - Very useful; 1- Not Useful at all Normal users ● SODALITE IDE : 4.56

	• YAML Editor: 3.56 TOSCA experts • SODALITE IDE : 4.2 • YAML Editor: 3.4 Use case owners • 4
Intention to use	Normal users • SODALITE IDE : 2.67 • YAML Editor: 1.44 TOSCA experts • SODALITE IDE : 2.7 • YAML Editor: 1.87 Use case owners • 2.5 1 to 3 scale where 3 - Intention to use ; 1- No intention to use

KPI 3.2	Reduction in software management (redeployment, reconfiguration) time and cost.
Target	Lower bound target is 30% improvement over the baseline and will be evaluated through external parties where possible. The improvement will be measured by changing the way we re-deploy the app.
Deadline	M24
Metric	**Value M24**
Time needed to redeploy and reconfigure an application.	TOSCA experts: • SODALITE IDE: 7.67 mins (average) • YAML EDITOR: 9.5 mins (average) **19.26 %** improvement over the baseline TOSCA experts modified the AADMs and TOSCA models of an application. Use Case Owners: • 37.5 mins (average) Use case owners extended and modified the AADMs of their use cases.

5.5 Platform Evaluation: Assessment of Integration KPIs (KPIs 4.1, 5.1, 5.2)

The assessment of KPIs 4.1, 5.1 and 5.2 has been conducted by examining the status of the components belonging to the SODALITE platform. The following tables provide an overview of the conducted assessment.

KPI 4.1	Component compatibility
Target	The target is 95% of SODALITE component compatibility
Deadline	M33
Metric	**Value M24**
# of components integrated in the SODALITE platform / total # of SODALITE components	25 components / 25 integrated components: 100% Note: the integration of some of the components will be further improved in the last project year.

KPI 5.1	Open source release
Target	Minimum 80% of code released under open-source license
Deadline	M36
Metric	**Value M24**
LOC released as open source / LOC produced by SODALITE to build the platform	100%*

*A version of MODAK has been included in the SODALITE GitHub repository, but continues to be developed internally by HPE. HPE is still in the process of determining whether future versions will continue to remain open.

KPI 5.2	Extension of existing projects
Target	Minimum 60% of code extending the existing projects, to be upstreamed
Deadline	M36
Metric	**Value M24**
LOC developed by SODALITE for a component and donated to OS / LOC developed by SODALITE for	96% of code submitted to upstream projects has been merged, a further 3% has been submitted but not yet merged.

the corresponding component	

5.6 Evaluation summary

The following table provides an overview on the level of accomplishment of all technical KPIs defined for the project:

Table 17 - Summary of technical KPI status at M24

KPI	Target	Due by	M24 status
KPI 1.1	25% coverage	M24	66% coverage
KPI 1.2	80% coverage	M33	Clinical UC: 80% coverage, Snow: 92% coverage, Vehicle IoT: TBM
KPI 1.3	80% coverage	M33	Clinical UC: 66% coverage, Snow: 83% coverage, Vehicle IoT: TBM
KPI 2.1	15% speedup increase	M30	Clinical UC: 3% speedup increase, Snow: 10% speedup increase
KPI 2.2	20% improvement over the baseline	M30	Node Manager: 96% reduction in SLA violations; 15% reduction in allocated resources Deployment refactoring: 96-99% accuracy and efficiency of performance prediction for deployment alternatives
KPI 3.1	10% improvement over the baseline	M24	28% improvement for a TOSCA expert
KPI 3.2	30% improvement over the baseline	M24	19% improvement for a TOSCA expert
KPI 4.1	95% component compatibility	M33	100% component compatibility, specific features to be improved
KPI 5.1	80% open source code	M36	100%* open source code
KPI 5.2	60% of upstreamed code	M36	96% of code submitted to upstream projects has been merged, a further 3% has been submitted but not yet merged.

*Refer to the note above about MODAK.

As can be seen, all KPIs, to the extent at which they can be measured, show a significantly high level of accomplishment, with the exception of KPI 3.2 which has suffered from some instability of the IDE at the time of the conducted usability experiments. All KPIs will be reassessed at the next project milestone.

Notably absent at this step is the performance measurement of the Vehicle IoT use case, which owing to its dependence on Kubernetes is not yet able to be measured or optimized directly by SODALITE components. Kubernetes support for the relevant SODALITE components is being first made available at the end of M24, meaning that integration, optimization, and continuous benchmarking can only begin in Y3. Performance baselines within the Vehicle IoT use case have already been defined at the component level, with optimization opportunities already identified.

As the work plan for the use case has allocated most of Y3's efforts in this area, this is more or less in line with the expected state of the project at this stage. The results from the other two use cases are, however, promising, and similar results are expected to be achieved in the Vehicle IoT case.

Independent of the technical KPIs, each of the demonstrating use cases are able to effectively leverage and benefit from the application of SODALITE components. The increase in SODALITE UML UC uptake is expected to continue in Y3.

6 Conclusions

This deliverable presents the Y2 iteration of the SODALITE platform, provides a comprehensive overview of the components and features developed throughout the year, the steps taken to ensure that components continue to maintain a high level of quality, and concludes with an in-depth look at the application of SODALITE components in each of the project's three demonstrating use cases.

Progress in the technical KPIs has shown that SODALITE is achieving its stated objectives and is producing clear benefits. The extent of integration and the uptake of the SODALITE UML UCs by the demonstrating use cases has shown that each of the use cases are able to make effective use of SODALITE components and realize direct benefits from their application. As the focus in Y3 shifts towards increased integration and optimization, the extent of benefits realized is expected to continue to increase.

The introduction of the Edge as a new infrastructure has created additional challenges for the project, while the Vehicle IoT use case's heavy use of Kubernetes in its operational environment has also created an opportunity for SODALITE to increase its interoperability with existing solutions, which will also be fundamental in facilitating uptake by users external to the consortium. As much of the enabling work at the component level has only just been completed in M24, this will be a key focus area going into Y3.

Feedback obtained from the External Advisory Board (detailed in D7.4) as well as from controlled experiments with use case owners and TOSCA experts alike has further highlighted areas for improvement in the final release. These aspects provide a clear outline of improvements to be carried out in Y3, and will be an important part of facilitating uptake of SODALITE results by external users, particularly by those with limited domain expertise.

This deliverable provides the second of three implementation and evaluation reports. The forthcoming D6.4 report will provide an assessment of the final release of the SODALITE platform and will provide an assessment of the Y3 items highlighted throughout this deliverable.

Reference

1. D6.2 - Initial implementation and evaluation of the SODALITE platform and use cases. SODALITE consortium. Public deliverable, 2020.
2. D6.6 SODALITE Framework - Second Version. SODALITE Technical report, 2021
3. D2.2 - Requirements, KPIs, evaluation plan and architecture - Intermediate version. SODALITE Technical Deliverable, 2021
4. D4.2 IaC Management - Intermediate version. SODALITE Technical Deliverable 2021.
5. D5.2 - Application deployment and dynamic runtime - intermediate version. SODALITE Technical Deliverable 2021.
6. https://github.com/OpenVPN/openvpn
7. https://github.com/traefik/traefik
8. https://github.com/SODALITE-EU/application-optimisation
9. https://github.com/SODALITE-EU/platform-discovery-service
10. https://github.com/cea-hpc/modules
11. https://www.cscs.ch/computers/piz-daint/
12. D3.3 - Prototype of application and infrastructure performance models - First version. SODALITE Technical Deliverable 2020.
13. https://www.egi.eu/
14. https://www.hpc-rivr.si/home_en/
15. https://www.egi.eu/services/applications-on-demand/
16. *On Understanding and Measuring the Performance of Public Cloud Providers*, Luciano Baresi, Giovanni Quattrocchi, Nicholas Rasi, Politecnico di Milano, 2020, http://hdl.handle.net/11311/1157381
17. https://semver.org/spec/v2.0.0.html - Semantic Versioning specification
18. https://github.com/marketplace/actions/release-drafter Drafts your next release notes as pull requests are merged into master.
19. https://github.com/marketplace/actions/pr-labeler - A GitHub Action that automatically applies labels to your PRs based on branch name patterns like feature/* or fix/*.
20. https://github.com/SODALITE-EU/iac-platform-stack
21. D2.4 - Guidelines for contributors to the SODALITE framework. SODALITE Deliverable 2020.
22. https://github.com/SODALITE-EU/iac-platform-stack
23. D6.1 - SODALITE platform and use cases implementation plan. SODALITE Technical Deliverable 2019.
24. D6.2 - SODALITE platform and use cases implementation plan. SODALITE Technical Deliverable 2020.
25. http://www.vtk.org
26. https://lorensen.github.io/VTKExamples/site/VTKFileFormats/
27. https://www.labri.fr/perso/pelegrin/scotch/
28. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview
29. https://www.openblas.net/
30. http://www.netlib.org/lapack/
31. http://www.netlib.org/scalapack/
32. http://www.netlib.org/blacs
33. http://mumps.enseeiht.fr/
34. https://portal.nersc.gov/project/sparse/superlu/
35. https://www.mcs.anl.gov/petsc/
36. https://www.salome-platform.org/user-section/about/med
37. https://www.knowgo.io
38. Deep residual learning for image recognition. He, Kaiming, et al. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.
39. Imagenet: A large-scale hierarchical image database, J. Deng et al., 2009 IEEE conference on computer vision and pattern recognition, 2009.
40. https://github.com/SODALITE-EU/iac-modules/tree/master/hpc
41. https://docs.egi.eu/users/cloud-compute/ec3/
42. https://github.com/SODALITE-EU/ide/tree/master/dsl/org.sodalite.dsl.examples
43. Deep Residual Learning for Image Recognition, K. He et al, arXiv:1512.03385 [cs.CV]
44. XLA: Optimizing Compiler for Machine Learning, https://www.tensorflow.org/xla, 2021.
45. Code_Aster: Finite element code aster, analysis of structures and thermomechanics for studies and research, http://www.code-aster.org, 2021.

46. Davis, Fred D. "Perceived usefulness, perceived ease of use, and user acceptance of information technology." MIS quarterly (1989): 319-340.