Sodalite

SOftware Defined AppLication Infrastructures managemenT and Engineering

SODALITE Framework - Second Version

D6.6

IBM and POLIMI

31.01.2021

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

Deliverable data			
Deliverable	D6.6 SODALITE framework - Second version		
Authors	Kalman Meth (IBM) Indika Kumara (JADS) Zoe Vasileiou, Vasileios-Rafahl Xefteris, Savvas Tzanakis, Anastasios Karakostas, Spyridon Symeonidis (CERTH) Jesús Gorroñogoitia, Lucas Pelegrin (ATOS) Giovanni Quattrocchi (POLIMI) Dragan Radolović, Nejc Bat (XLAB) Alfio Lazzaro (HPE) Saloni Kyal (POLIMI) Paul Mundt (ADPT)		
Reviewers	Nejc Bat (XLAB) Kamil Tokmakov (USTUTT)		
Dissemination level	Public, DEM		
History of changes	Name	Change	Date
	Kalman Meth (IBM)	Outline created	31.10.2020
	All	Components Info updated; Improved features described; Introduction/conclusion updated	13.01.2021
	All	final release	28.01.2021

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

Table of Contents **2**

List Of Images **9**

List Of Tables **9**

Executive Summary **11**

Glossary **12**

1 Introduction **13**

1.1 Structure of the Document 14

1.2 SODALITE Context and Goals 14

1.3 Overview of SODALITE architecture 15

1.3.1 Modelling Layer 15

1.3.2 Infrastructure as Code (IaC) Layer 16

1.3.3 Runtime Layer 17

2 Improved Integrated Prototype - Milestone 6 **18**

2.1 Integrated Prototype Improved Features 20

2.2 SODALITE Security Support 20

2.2.1 IAM Authentication and Introspection 21

2.2.2 Secrets Management 21

2.3 Deploying the Integrated Prototype 21

2.3.1 Deploying the Sodalite Stack via Common blueprint 21

2.3.2 Deploying the IDE 21

2.4 Running the Integrated Prototype 21

2.5 Supported Infrastructures 22

2.6 Common development/testing tools/methods used in the project 22

2.7 How can an external user exploit Sodalite? 22

2.7.1 Tutorial for using the SODALITE platform 22

3 Modelling Layer **23**

3.1 Semantic Knowledge Base 23

3.1.1 Description of component 23

3.1.2 Status of implementation 23

3.1.3 Location of repository and how to build the code 23

3.1.4 Dependency of component on other components and stand-alone usage of component 23

3.2 Semantic Reasoner 23

3.2.1 Description of component 23

3.2.2 Status of implementation 24

3.2.3 Location of repository and how to build the code 24

3.2.4 Dependency of component on other components and stand-alone usage of component 24

3.3 SODALITE IDE 24
3.3.1 Description of component 24
3.3.2 Status of implementation 25
3.3.3 Location of repository and how to build the code 25
3.3.4 Dependency of component on other components and stand-alone usage of component 25

4 IaC Management Components 26
4.1 Abstract Model Parser 26
4.1.1 Description of component 26
4.1.2 Status of implementation 26
4.1.3 Location of repository and how to build the code 26
4.1.4 Dependency of component on other components and stand-alone usage of component 26
4.2 IaC Blueprint Builder 26
4.2.1 Description of component 26
4.2.2 Status of implementation 26
4.2.3 Location of repository and how to build the code 26
4.2.4 Dependency of component on other components and stand-alone usage of component 27
4.3 Runtime Image Builder 27
4.3.1 Description of component 27
4.3.2 Status of implementation 27
4.3.3 Location of repository and how to build the code 27
4.3.4 Dependency of component on other components and stand-alone usage of component 27
4.4 Concrete Image Builder 27
4.4.1 Description of component 27
4.4.2 Status of implementation 27
4.4.3 Location of repository and how to build the code 28
4.4.4 Dependency of component on other components and stand-alone usage of component 28
4.5 Image Registry 28
4.5.1 Description of component 28
4.5.2 Status of implementation 28
4.5.3 Location of repository and how to build the code 28
4.5.4 Dependency of component on other components and stand-alone usage of component 28
4.6 Application Optimiser 28
4.6.1 Description of component 28
4.6.2 Status of implementation 28
4.6.3 Location of repository and how to build the code 28
4.6.4 Dependency of component on other components and stand-alone usage of component 28

4.7 IaC Verifier 29
4.7.1 Description of component 29
4.7.2 Status of implementation 29
4.7.3 Location of repository and how to build the code 29
4.7.4 Dependency of component on other components and stand-alone usage of component 29
4.8 Verification Model Builder 29
4.8.1 Description of component 29
4.8.2 Status of implementation 29
4.8.3 Location of repository and how to build the code 29
4.8.4 Dependency of component on other components and stand-alone usage of component 29
4.9 Topology Verifier 29
4.9.1 Description of component 29
4.9.2 Status of implementation 30
4.9.3 Location of repository and how to build the code 30
4.9.4 Dependency of component on other components and stand-alone usage of component 30
4.10 Provisioning Workflow Verifier 30
4.10.1 Description of component 30
4.10.2 Status of implementation 30
4.10.3 Location of repository and how to build the code 30
4.10.4 Dependency of component on other components and stand-alone usage of component 30
4.11 Bug Predictor and Fixer 30
4.11.1 Description of component 30
4.11.2 Status of implementation 30
4.11.3 Location of repository and how to build the code 31
4.11.4 Dependency of component on other components and stand-alone usage of component 31
4.12 Predictive Model Builder 31
4.12.1 Description of component 31
4.12.2 Status of implementation 31
4.12.3 Location of repository and how to build the code 31
4.12.4 Dependency of component on other components and stand-alone usage of component 31
4.13 IaC Quality Assessor 31
4.13.1 Description of component 31
4.13.2 Status of implementation 31
4.13.3 Location of repository and how to build the code 31
4.13.4 Dependency of component on other components and stand-alone usage of component 31
4.14 Platform Discovery Service 31
4.14.1 Description of component 31

4.14.2 Status of implementation 32
4.14.3 Location of repository and how to build the code 32
4.14.4 Dependency of component on other components and stand-alone usage of component 32
4.15 IaC Model Repository 32
4.15.1 Description of component 32
4.15.2 Status of implementation 32
4.15.3 Location of repository and how to build the code 32
4.15.4 Dependency of component on other components and stand-alone usage of component 32

5 Runtime Layer Components 33
5.1 Orchestrator - xOpera 33
5.1.1 Description of component 33
5.1.2 Status of implementation 33
5.1.3 Location of repository and how to build the code 33
5.1.4 Dependency of component on other components and stand-alone usage of component 33
5.2 xOpera REST API 33
5.2.1 Description of component 33
5.2.2 Status of implementation 33
5.2.3 Location of repository and how to build the code 33
5.2.4 Dependency of component on other components and stand-alone usage of component 33
5.3 Deployment Refactorer 34
5.3.1 Description of component 34
5.3.2 Status of implementation 34
5.3.3 Location of repository and how to build the code 34
5.3.4 Dependency of component on other components and stand-alone usage of component 34
5.4 Node Manager 34
5.4.1 Description of component 34
5.4.2 Status of implementation 34
5.4.3 Location of repository and how to build the code 35
5.4.4 Dependency of component on other components and stand-alone usage of component 35
5.5 Refactoring Option Discoverer 35
5.5.1 Description of component 35
5.5.2 Status of implementation 35
5.5.3 Location of repository and how to build the code 35
5.5.4 Dependency of component on other components and stand-alone usage of component 35
5.6 ALDE 35
5.6.1 Description of component 35

5.6.2 Status of implementation 35
5.6.3 Location of repository and how to build the code 35
5.6.4 Dependency of component on other components and stand-alone usage of component 35
5.7 Monitoring - Prometheus/Consul 36
5.7.1 Description of component 36
5.7.2 Status of implementation 36
5.7.3 Location of repository and how to build the code 36
5.7.4 Dependency of component on other components and stand-alone usage of component 36
5.8 Alert Manager/Rule File Server 36
5.8.1 Description of component 36
5.8.2 Status of implementation 36
5.8.3 Location of repository and how to build the code 36
5.8.4 Dependency of component on other components and stand-alone usage of component 36
5.9 Node Exporter 37
5.9.1 Description of component 37
5.9.2 Status of implementation 37
5.9.3 Location of repository and how to build the code 37
5.9.4 Dependency of component on other components and stand-alone usage of component 37
5.10 IPMI Exporter 37
5.10.1 Description of component 37
5.10.2 Status of implementation 37
5.10.3 Location of repository and how to build the code 37
5.10.4 Dependency of component on other components and stand-alone usage of component 37
5.11 HPC Exporter 37
5.11.1 Description of component 37
5.11.2 Status of implementation 37
5.11.3 Location of repository and how to build the code 38
5.11.4 Dependency of component on other components and stand-alone usage of component 38
5.12 Skydive 38
5.12.1 Description of component 38
5.12.2 Status of implementation 38
5.12.3 Location of repository and how to build the code 38
5.12.4 Dependency of component on other components and stand-alone usage of component 38
5.13 Edge Exporter 39
5.13.1 Description of component 39
5.13.2 Status of implementation 39
5.13.3 Location of repository and how to build the code 39

5.13.4 Dependency of component on other components and stand-alone usage of component 39

6 Conclusion 40

7 References 41

Appendix A - Reference Component Definition 42

Appendix A.1 SODALITE Semantic Modelling Layer 42

A.1.1 SODALITE IDE 42

A.1.2 Semantic Reasoner 44

A.1.3 Semantic Knowledge Base 45

A.1.4 Development plan for Semantic Modelling Layer 46

Appendix A.2 SODALITE Infrastructure as Code Layer 47

A.2.1 Abstract Model Parser 47

A.2.2 IaC Blueprint Builder 48

A.2.3 Runtime Image Builder 48

A.2.4 Concrete Image Builder 50

A.2.5 Application Optimiser 50

A.2.6 IaC Verifier 51

A.2.7 Verification Model Builder 52

A.2.8 Topology Verifier 52

A.2.9 Provisioning Workflow Verifier 53

A.2.10 Bug Predictor and Fixer 53

A.2.11 Predictive Model Builder 54

A.2.12 IaC Quality Assessor 55

A.2.13 IaC Model Repository 55

A.2.14 Image Registry 56

A.2.15 Platform Discovery Service 57

A.2.16 Development plan for Infrastructure as Code Management layer 58

Appendix A.3 SODALITE Runtime Layer 60

A.3.1 Orchestrator + Drivers 60

Orchestrator 60

ALDE 61

A.3.2 Monitoring 62

Monitoring system 62

Alert Manager 63

Monitoring Dashboard - Grafana 64

IPMI Exporter 64

HPC Exporter 65

Skydive Exporter 65

A.3.3 Deployment Refactorer 66

A.3.4 Node Manager 67

A.3.5 Refactoring Option Discoverer 68

A.3.6 xOpera REST API 68
A.3.7. Kubernetes Edge Components 69
Controllers / Monitors 69
Labellers 70
Device Plugins 71
Exporters 72
A.3.7 Development plan for Runtime Layer 73
Deployment Refactorer 74
Node Manager 75
Refactoring Option Discoverer 75
xOpera REST API 75
Kubernetes Controller for Edge 76
Appendix A.4 SODALITE Security Components 76
A.4.1 IAM Introspection 76
A.4.2 Secrets Management 77

List Of Images

- Figure 1 - SODALITE Overall Architecture.
- Figure 2 - Modelling Layer Architecture.
- Figure 3 - Updated Infrastructure as Code Layer Architecture.
- Figure 4 - Runtime Layer Architecture.
- Figure 5 - NCS2 Exporter instrumentation requirements

List Of Tables

- Table 1 - Development status of the Second Prototype
- Table 2 - Development status of the IDE
- Table 3 - Development status of Semantic Reasoning Engine
- Table 4 - Development status of Semantic Population Engine
- Table 5 - Development status of RDF Triple Store
- Table 6 - Development status of Domain Ontologies
- Table 7 - Development plan for Semantic Modelling Layer
- Table 8 - Development status of Abstract Model Parser
- Table 9 - Development status of IaC Blueprint Builder
- Table 10 - Development status of Runtime Image Builder
- Table 11 - Development status of Concrete Image Builder
- Table 12 - Development status of Application Optimiser - MODAK
- Table 13 - Development status of IaC Verifier
- Table 14 - Development status of Verification Model Builder
- Table 15 - Development status of Topology Verifier
- Table 16 - Development status of Deployment Exporter
- Table 17 - Development status of Bug Predictor and Fixer
- Table 18 - Development status of Predictive Model Builder
- Table 19 - Development status of IaC Quality Assessor
- Table 20 - Development status of IaC Model Repository
- Table 21 - Development status of Image Registry
- Table 22 - Development status of Platform Discovery Service
- Table 23 - Development plan for IaC Management Layer
- Table 24 - Development status of Orchestrator - xOpera
- Table 25 - Development status of ALDE
- Table 26 - Development status of Monitoring - Prometheus
- Table 27 - Development status of Monitoring - Consul
- Table 28 - Development status of Alert Manager
- Table 29 - Development status of Monitoring Dashboard
- Table 30 - Development status of IPMI Exporter
- Table 31 - Development status of HPC Exporter
- Table 32 - Development status of Skydive Exporter
- Table 33 - Development status of Deployment Refactorer
- Table 34 - Development status of Node Manager
- Table 35 - Development status of Refactoring Option Discoverer

- Table 36 - Development status of xOpera REST API
- Table 37 - Development status of K8s Node Label Monitor
- Table 38 - Development status of K8s DeviceTree Node Labeller
- Table 39 - Development status of K8s Node Auto Labeller
- Table 40 - Development status of Intel NCS2 Device Plugin
- Table 41 - Development status of Intel NCS2 Exporter
- Table 42 - Development status of Google Coral EdgeTPU Exporter
- Table 43 - Development plan for Runtime Layer
- Table 44 - Development status of IAM Introspection
- Table 45 - Development status of Secrets Management

Executive Summary

The SODALITE Framework is the software system that includes all SODALITE stable components. This deliverable includes the description of the software that makes up the SODALITE stack, while the actual software is available through the GitHub SODALITE repository (https://github.com/SODALITE-EU). The document thus serves as an accompanying textual document, describing the components delivered by the SODALITE consortium at the end of the second year of the project. This document updates the status reported in D6.5 - *SODALITE Framework - First Version*. This comprises what has been achieved for the second SODALITE prototype embodied in Milestone 6 (MS6) of the project which was defined as: First advanced features, more integrated prototype running. Use-Cases are clearly improved. Second public release of the complete stack.

The SODALITE architecture is divided into 3 main layers: Modelling Layer, Infrastructure as Code Layer and Runtime Layer. This document summarizes the available stable components in each of these layers, and points to instructions on how these components can be accessed and built.

In addition to the updated status of each of the components, this document adds a section (section 2) outlining the improvements in this release of the SODALITE stack.

This document represents the status at the end of Year 2 of the project and will be updated with a future release of the framework at the end of Year 3 (D6.7 - *SODALITE framework - Final version*).

Glossary

Acronym	Explanation
AADM	Abstract Application Deployment Model
AAI	Authentication and Authorisation Infrastructure
ALDE	Application Lifecycle Deploy Engine
AM	Ansible Model
AOE	Application Ops Experts
API	Application Program Interface
CSAR	Cloud Service Archive
DSL	Domain Specific Language
HPC	High Performance Computing
IaC	Infrastructure as Code
IAM	Identity and Access Management
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
KB	Knowledge Base
KPI	Key Performance Indicator
MODAK	Model based Optimiser for Deployment of Application (with Kontainers)
NLP	Natural Language Processing
OM	Optimisation Model
OSS	Open Source Software
RDF	Resource Description Framework
REST	Representational State Transfer
RM	Resource Model
TOSCA	Topology and Orchestration Specification for Cloud Applications
UC	Use Case

1 Introduction

The SODALITE Framework is the software system that includes all SODALITE stable components. **This deliverable presents the status and location of the software that make up the SODALITE stack, describing the components delivered by the SODALITE consortium at the end of the second year of the project.** This document updates the status reported in D6.5 - *SODALITE Framework - First Version*. This is mainly a software deliverable (The deliverable is marked as DEM - it contains software accompanying information.) and comprises what has been achieved for the SODALITE prototype embodied in Milestone 6 (MS6) of the project which was defined as: *First advanced features, more integrated prototype running. Use-Cases are clearly improved. Second public release of the complete stack.*

All of the released components of this Second Prototype use a CI/CD pipeline to automatically build their artifacts. As part of the build process, individual components are packaged in Docker containers and are stored in the SODALITE DockerHub repository[1]. A blueprint[2] was prepared that allows deployment of the entire stack using the xOpera orchestrator. In addition, instructions are provided individually for each component on how to compile and use the component.

Instructions on how to contribute to the SODALITE stack are provided in deliverable D2.4, *Guidelines for contributors to the SODALITE framework*[3].

The SODALITE architecture is divided into 3 main layers: Modelling Layer, Infrastructure as Code Layer and Runtime Layer. A laboratory prototype of the SODALITE Framework is running on the SODALITE testbed, and the SODALITE Use-Cases run on the prototype. This document summarizes the available stable components in each of the layers, and points to instructions on how they can be accessed and built.

Video demonstrations of many of these components are presented on the Project's YouTube channel[4].

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

In addition to the updated status of each of the components reported in D6.5, this document adds a section (section 2) outlining the improvements in this release of the SODALITE stack.

This document represents the status at the end of Year 2 of the project and will be updated with a future release of the framework at the end of Year 3 (D6.7 - *SODALITE framework - Final version*).

1.1 Structure of the Document

The next subsections (1.2 and 1.3) reproduce (mainly from D2.1[5] - *Requirements, KPIs, evaluation plan and architecture - First version* and D2.2 - *Requirements, KPIs, evaluation plan and architecture - Intermediate version*) an overview of the SODALITE Context and Goals followed by the SODALITE architecture and components that make up the SODALITE Framework. This is followed by section 2, which outlines the improvements that were added to this second prototype release. The following sections of the document (sections 3, 4, 5) list the components (grouped according to the architecture structure) that make up the SODALITE Platform. For each component, we provide a short description, status, and pointers to source code repository and instructions on how to build the code. These status reports update the previous version of this deliverable, D6.5 (*SODALITE Framework - First Version*). In addition, the component information is presented in compact, tabular form in an appendix.

1.2 SODALITE Context and Goals

The SODALITE vision is to *support Digital Transformation of European Industry through (1) increasing design and runtime effectiveness of software-defined infrastructures, to ensure high-performance execution over dynamic heterogeneous execution environments; (2) increasing simplicity of modelling applications and infrastructures, to improve manageability, collaboration, and time to market*.

Within this vision, SODALITE *provides application developers and infrastructure operators with tools that (a) abstract their application and infrastructure requirements to (b) enable simpler and faster development, deployment, operation, and execution of heterogeneous applications reflecting diverse circumstances over (c) heterogeneous, software-defined, high-performance, cloud infrastructures, with a particular focus on performance, quality, manageability, and reliability*.

In particular, SODALITE is focusing on supporting the entire life cycle of the so-called Infrastructure as Code (IaC). IaC means limiting the need to manually provision resources, configuring them and deploying an application by offering to DevOps teams the possibility to code such tasks into proper scripts that are then executed by proper orchestrators, thus introducing significant automation in the application life cycle.

The following innovations are offered:

1. Application Deployment Modeling and Infrastructure as Code Modeling - Build deployment patterns based on preexisting models, including rule-based models, semantic models, and data-driven (machine learning and deep learning).
2. Ease of Deployment, Orchestration, and Provisioning.
3. Verification and Bug Prediction - Support for a subset of smells, anti-patterns, and bugs in IaC scripts (TOSCA and Ansible).
4. Monitoring and Reconfiguration - Monitoring of metrics relevant for Cloud, HPC, and Edge computing environments and applications; Basic event-driven deployment refactoring decision making; Support for detecting and fixing performance anti-patterns.
5. Performance Optimisation - Static application optimisation for Cloud, HPC and Edge; Support for modelling performance.

1.3 Overview of SODALITE architecture

We reproduce here the figures and overview of the SODALITE architecture for simple reference. Additional details of the figures can be found in the Architecture sections of D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version*) and D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version*).

The SODALITE platform is divided into three main layers. These layers are the Modelling Layer, the Infrastructure as Code Layer, and the Runtime Layer. Figure 1 shows these layers together with their relationships defined in terms of interfaces. The Modelling Layer exploits the interfaces offered by the other two layers to offer to the end users (Application Ops Experts, Resource Experts and Quality Experts) the needed information concerning the application deployment configuration and the corresponding runtime. In turn, it offers to the other layers the possibility to access the ontology and the application deployment model through the SemanticReasoningAPI. The Infrastructure as Code Layer offers to the Modelling Layer the APIs for preparing the deployment, for verifying the IaC and for predicting defects. Finally, the Runtime Layer offers APIs for controlling the orchestration of an application deployment and for monitoring the status of the system. In turn, this layer relies on the interfaces offered by the underlying technologies with particular reference to the ones shown in the figure.

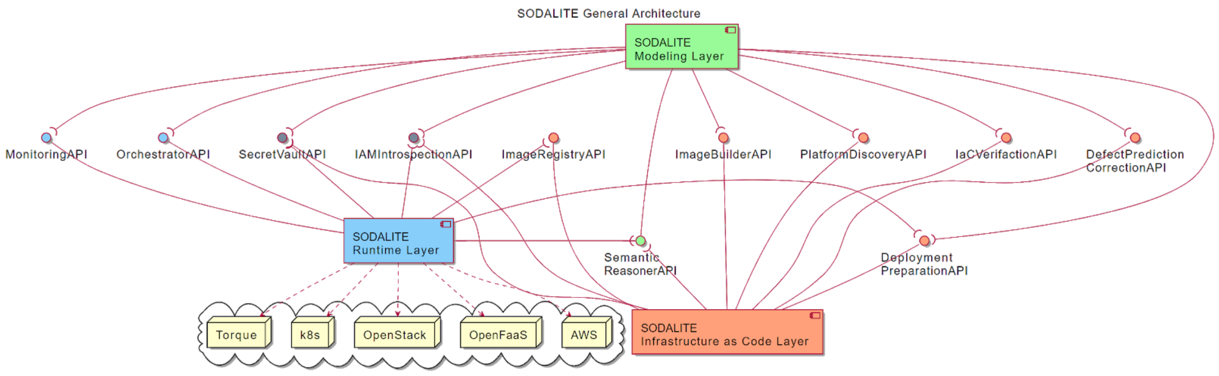

Figure 1 - SODALITE Overall Architecture.

1.3.1 Modelling Layer

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

The SODALITE IDE provides complete support for the authoring lifecycle of abstract application deployment models. The Semantic Knowledge Base (KB) is SODALITE's semantic repository that hosts the models (ontologies). The Semantic Reasoner is a middleware facilitating the interaction with the KB. In particular, it provides an API to support the insertion and retrieval of knowledge to/from the KB, and the application of rule-based semantic reasoning over the data stored in the KB. It mainly interacts with IDE, but also with various components of the IaC layer such as Platform Discovery Service, and Bug Predictor. During the second year, a part of the architecture was redesigned, depicted in Figure 2, and more details are reported in D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version*).

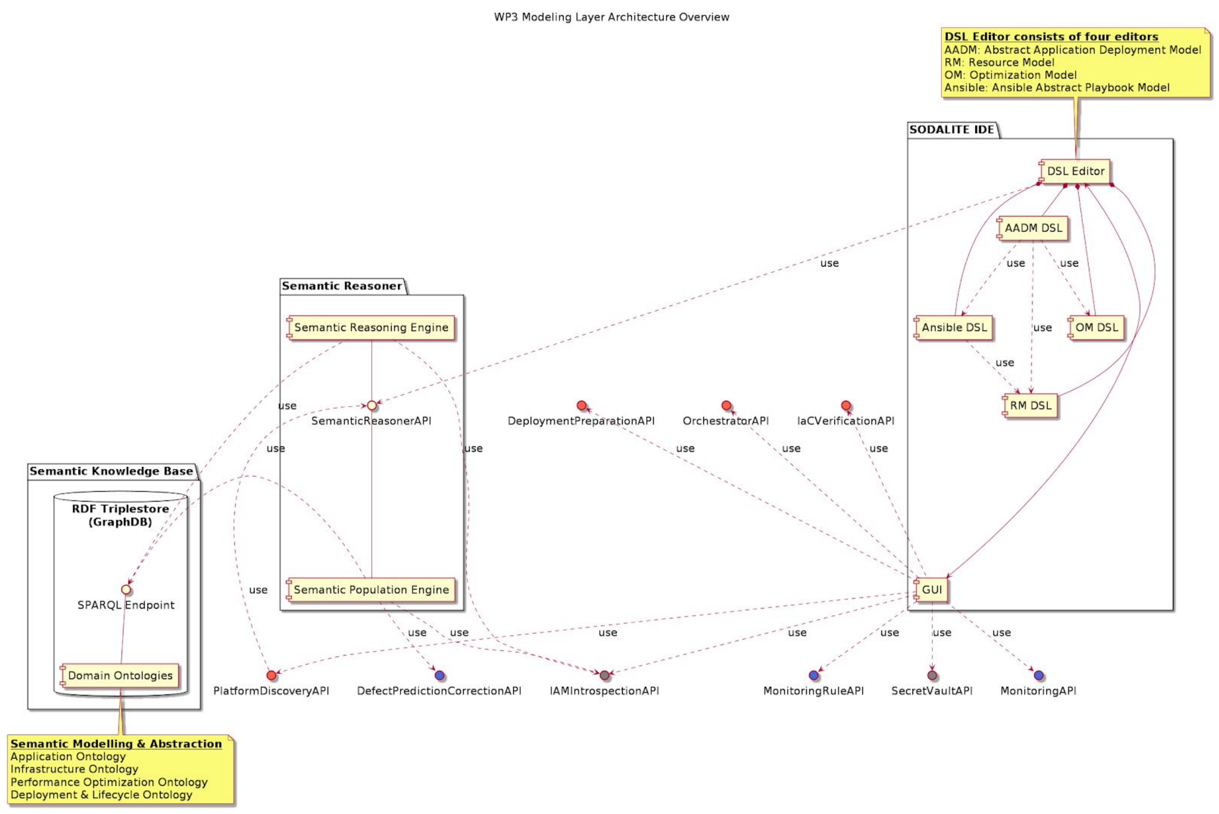

Figure 2 - Modelling Layer Architecture.

SODALITE IDE goes beyond existing approaches, by extending the modeling support to both Cloud and HPC domains, and by incorporating specific modeling assistance for the optimisation of application deployments.

Beyond providing a simplified DSL for Resource Model authoring, SODALITE releases the Resource Experts and Application Ops Experts from the complexity of the AADM and TOSCA YAML specification.

Additional details can be found in deliverable D2.2 and D3.1[6].

1.3.2 Infrastructure as Code (IaC) Layer

Preparing a valid and deployable TOSCA blueprint is one of the main tasks of IaC Layer. During the second year of the project several components were initially released, others refactored.

In this period Application Optimiser component exposing a dockerized REST API (MODAK) was initially released and integrated into the pipeline enabling the SODALITE users to statically optimise the application for a given target execution platform. Automation of application optimisation on both HPC and cloud systems requiring models that can be used for performance prediction has been improved. SODALITE prepares and uses these models for both pre-deployment (static) performance optimisation and runtime (dynamic) performance optimisation.

Platform Discovery Service was introduced in the architecture, exposing a REST API that helps to automate the tasks of the Resource Expert by creating a valid TOSCA platform resource model to be stored into the SODALITE's Knowledge Base and used during the design of the application deployment model (AADM).

During development in the second year of the project a part of the architecture was redesigned which was also reported in deliverable D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version*) and D4.2 (*IaC Management - Intermediate version*), shown here in Figure 3.

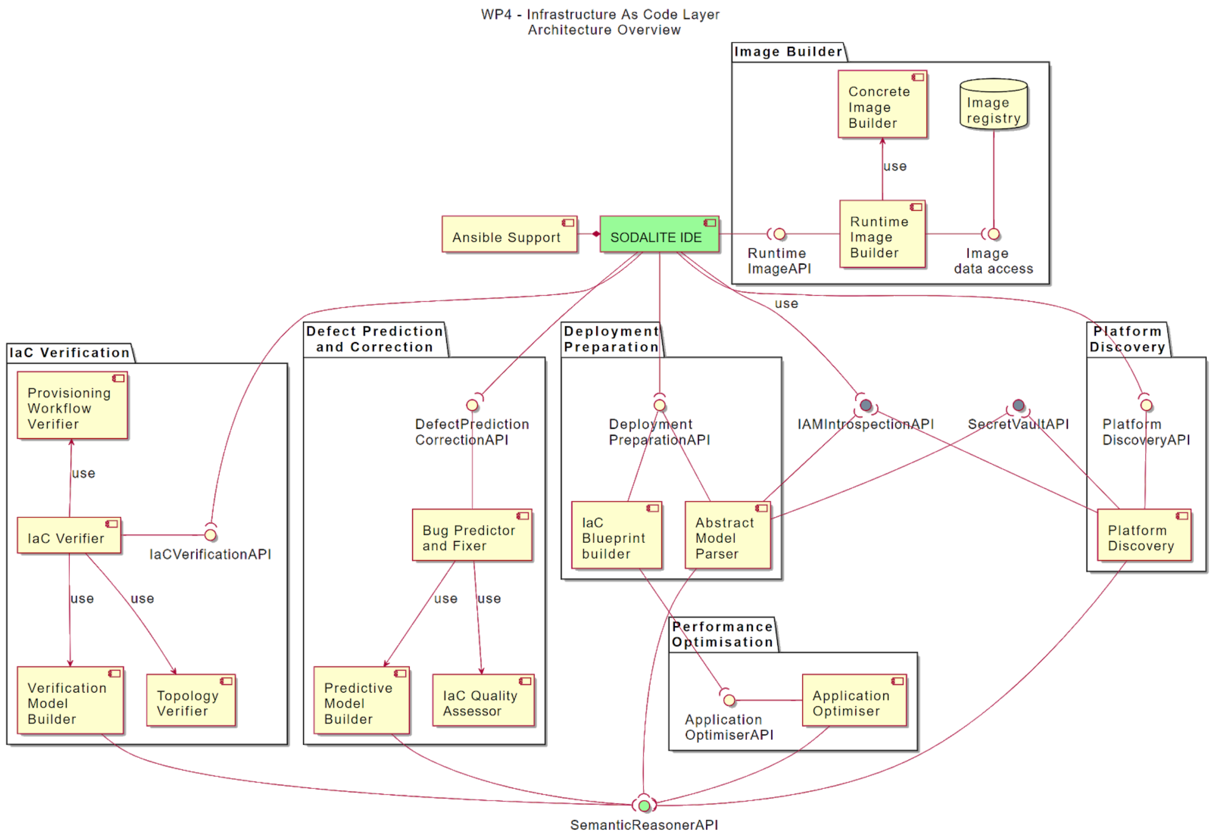

Figure 3 - Updated Infrastructure as Code Layer Architecture.

SODALITE's deployment preparation uses target platform optimised containers for execution on heterogeneous environments ranging from edge devices to private/ public clouds and HPC clusters.

An important step for producing a bug and code-smell free IaC builds on using a bug and smell IaC taxonomy for cloud and HPC applications based on a systematic literature review and a qualitative analysis of bug fix commit messages. SODALITE uses semantic technologies for verifying structural constraints and semantics of IaC, and also support for explaining errors and their causes, recommending the resolutions, (semi) automating the correction of erroneous IaC code through model-to-model transformations.

For additional details please refer to deliverable D4.2 (*IaC Management - Intermediate version*.)

1.3.3 Runtime Layer

The Runtime Layer of SODALITE, shown in Figure 4, orchestrates the deployment of an application, monitors its execution and proposes changes to the application's runtime. It is composed of three main blocks: Orchestrator, Monitoring and Refactoring. The Orchestrator manages the lifecycle of an application deployed in heterogeneous infrastructures. The Monitoring component gathers metrics from the heterogeneous infrastructures. These metrics are used to determine to what extent the application is running as expected. The Deployment Refactorer refactors the deployment model of an application in response to violations in the application goals.

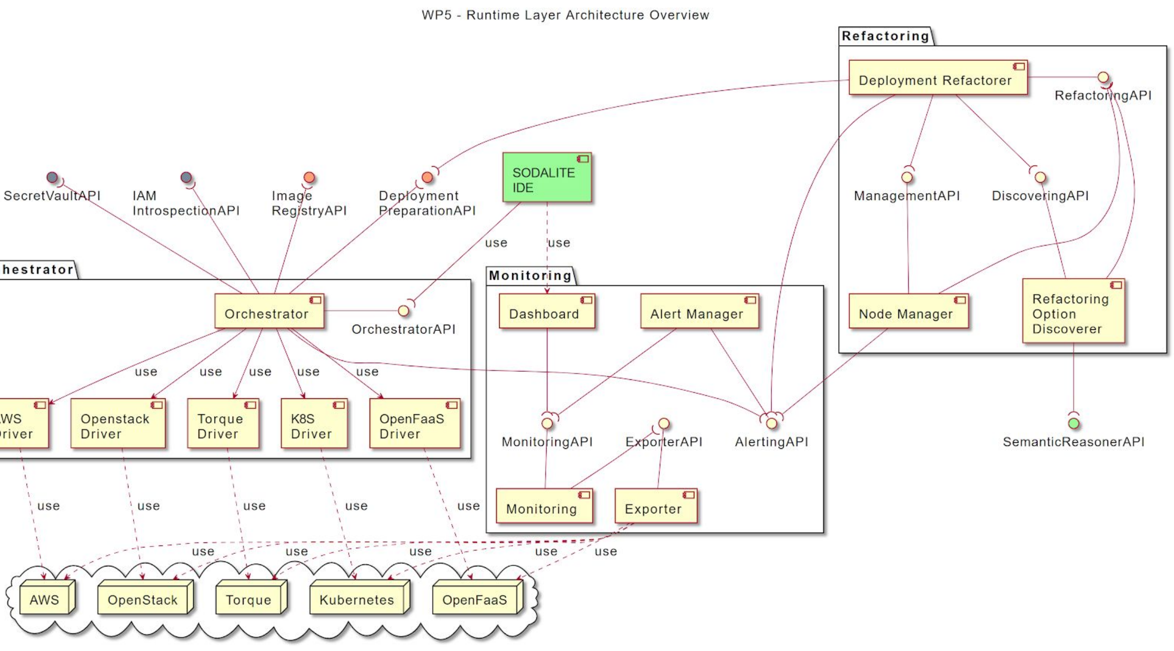

Figure 4 - Runtime Layer Architecture.

The SODALITE Orchestrator differs from other approaches that follow an intrusive architecture that require modifications to the infrastructure configuration. Our approach is to orchestrate resources via the existing resource managers and execution platforms.

Monitoring supports the dynamic setup of standard and specialized exporters for Cloud infrastructures, as well as for HPC and Edge. Real-time dynamic alerting rules and notifications to subscribers are supported as well. Standard and specialized reports for end-users are accessible through the Monitoring dashboard.

We have a novel approach for the deployment optimisation problem by combining rule-based, machine-learning based, and control-theory based techniques. We adopt the dynamic software product lines view, finding the optimal deployment variant between a set of allowed variants. To this end, we develop machine-learning based predictors that accurately predict performance of the whole population of the deployment variants based on a small sample of measured deployment variants. This allowed set of deployment variants is evolved at runtime by discovering and integrating new resources, nodes, and components. Moreover, we develop control-theory based planners to support fast vertical elasticity for containerized applications. We also aim to detect performance anti-patterns, being able to correct them in application deployments at runtime.

Additional details can be found in deliverable D5.2 (*Application deployment and dynamic runtime - intermediate version*).

2 Improved Integrated Prototype - Milestone 6

This section describes the development status of the MS6 - Second Prototype and its constituent components and modules. The table below 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 refers to whether 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 refers to whether the component is deployed via IaC on a particular infrastructure, e.g. Cloud testbed.
- Integration status refers to whether 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 past year, the development of all the components have progressed and new or advanced features have been implemented. New components have been added to further enhance the SODALITE platform, such as MODAK (Application Optimiser) and Platform Discovery Service. Furthermore, new Identity and Access Management (IAM) components were introduced into the overall SODALITE architecture. The integration of tools for Software Quality measurement helped to continuously estimate the quality of the developments.

The integration work was significantly improved thanks to the improved CI/CD, agreed conversion for release management and an introduction of SODALITE IaC Platform Stack[7]. New components are partially integrated, such as MODAK, Platform Discovery and IAM.

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**

Table 1 - Development status of the Second Prototype

Later sections provide details of the development status of the components of each layer of the Second Prototype. Additional information is presented in an appendix in tabular form including pointers to the source code and downloadable artifacts, dependencies and steps towards the next prototype.

2.1 Integrated Prototype Improved Features

The following improvements are included in the updated SODALITE platform.

- Modelling Layer
    - Support of TOSCA Policies in the Modelling Layer.
    - Improved context assistance in model authoring.
    - Improved scalability in the semantic services.
- IaC layer
    - Automated discovery and TOSCA description of infrastructure.
    - Support for the creation of Ansible scripts integrated with the Resource Models.
    - Static optimisation of applications before deployment (MODAK).
    - Semantic and Analysis Support including: Bug Taxonomy; Unified best and bad practices Catalog; Unified Smell Catalog; Linguistic Anti-pattern detection via NLP and Deep Learning.
    - Orchestration: Improvements to xOpera including Reconfiguration; Parallel execution of deployment; Resume command; Worklows; REST API.
- Runtime Layer
    - Dynamic Monitoring.
    - Refactoring.

Details of these features can be found in deliverable D4.2 (*IaC Management - Intermediate version*) and deliverable D5.2 (*Application deployment and dynamic runtime - intermediate version*).

2.2 SODALITE Security Support

SODALITE provides tools and methods to authenticate and authorize actions on SODALITE's API endpoints using open-source tools. Authorization is required, since SODALITE endpoints can manage different infrastructures belonging to different domains. Authentication and authorization is implemented using IAM (Identity and Access Management) authentication endpoint and introspection API endpoints. Apart from proper authorization of users actions, safe secret management across the whole deployment pipeline is also required and ensured by SODALITE using Secret Vault endpoint which is also integrated into the SODALITE integrated prototype and used in the deployment workflows.

2.2.1 IAM Authentication and Introspection

As a basis for authorization the OAuth 2.0 protocol was chosen, which is a de-facto industry standard for authorization. As for IAM provider, SODALITE uses Keycloak[8] - a popular open source tool which simplifies the creation of secure services with minimal coding for authentication and authorization. Keycloak covers all the security IAM workflows needed by SODALITE.

2.2.2 Secrets Management

Apart from properly authorizing users actions, other concerns are also addressed - properly handling infrastructure secrets, like RSA keys, tokens, passwords. For that purpose Hashicorp Vault[9] was chosen, which is one of the most widely used open source tools for secret management. This approach allows SODALITE operators to integrate it with their own Vault installations that are not part of the SODALITE stack.

2.3 Deploying the Integrated Prototype

SODALITE approaches the deployment of the framework prototype by using a TOSCA blueprint and xOpera orchestrator. The deployment is further simplified by running a script, which installs a local copy of the xOpera orchestrator and all the needed packages, and configures the deployment of the SODALITE stack to different targets (local Ubuntu machine, OpenStack VM instance). The script finally executes the deployment of the TOSCA blueprint, which describes the SODALITE model and deploys the SODALITE dockerized components from the public DockerHub repositories, using xOpera orchestrator.

2.3.1 Deploying the Sodalite Stack via Common blueprint

SODALITE deployment blueprints with all information needed to deploy SODALITE stack are defined in the GitHub repository https://github.com/SODALITE-EU/iac-platform-stack, offering extensive descriptions on how to configure all the components.

2.3.2 Deploying the IDE

The SODALITE IDE component should be installed separately from the SODALITE IDE GitHub repository https://github.com/SODALITE-EU/ide, offering extensive information on how to install and configure the SODALITE IDE.

2.4 Running the Integrated Prototype

After the installation of the SODALITE backend components via the TOSCA blueprint deployment and the installation and configuration of the SODALITE IDE the user is able to start using the SODALITE stack. A simple example model used for test runs can be found in the SODALITE IDE repository under:
https://github.com/SODALITE-EU/ide/tree/master/dsl/org.sodalite.dsl.examples/sodalite-test,

while other, somewhat more complex examples can be found in this part of the SODALITE IDE repository:
https://github.com/SODALITE-EU/ide/tree/master/dsl/org.sodalite.dsl.examples

NOTE: Some of the examples need specific Openstack/HPC (Torque/Slurm) infrastructure access.

2.5 Supported Infrastructures

Currently, SODALITE supports application deployment on heterogeneous platform infrastructures such as HPC clusters managed by Torque and Slurm resource managers, Openstack private/public cloud, public AWS EC2, and Kubernetes on Edge through Helm chart deployments.

2.6 Common development/testing tools/methods used in the project

The project uses the **GitHub** platform for code management (https://github.com/SODALITE-EU), 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** (https://www.jenkins.io/) platform was chosen to conduct the CI/CD operations for the project's components. Jenkins provides all the functionality we need to perform CI/CD, is considered one of the most popular CI/CD platforms today, and is also available as OSS.

DockerHub (https://hub.docker.com/) 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 a high quality, it was decided to use the **SonarCloud** (https://sonarcloud.io/) platform for static code analysis and code coverage testing.

Additional details can be found in D6.3 (*Intermediate Implementation and Evaluation of the SODALITE Platform and Use Cases*).

2.7 How can an external user exploit Sodalite?

2.7.1 Tutorial for using the SODALITE platform

A tutorial[10] that explains how AOEs can create AADMs is available. This tutorial has been used as documentation for the participants in a SODALITE workshop. In new releases, this tutorial will be extended to cover other IDE features, including AADM deployment or RM authoring.

Video demonstrations of many of these components are presented on the Project's YouTube channel (https://www.youtube.com/channel/UCrArVp55GaJs78jFt1lUfFg).

3 Modelling Layer

This section lists the components of the Modelling Layer, describing each component's function, implementation status, and code repository. This section updates the component status reported in D6.5 (*SODALITE Framework - First Version*). A concise list of improvements of the current release over the previous release appears in section 2.1.

3.1 Semantic Knowledge Base

3.1.1 Description of component

The Semantic Knowledge Base (KB) is the semantic database management system of SODALITE that enables storing, querying and managing structured data. It follows the semantic data schema paradigm, called ontology, which is stored and managed independently from the data (see D3.1, *First version of ontologies and semantic repository*, for more details).

3.1.2 Status of implementation

KB uses an existing RDF triple store (GraphDB[11]) to persist and index the ontologies developed regarding the modelling of applications and resources. The configuration of the KB has been tuned so as to achieve better performance. The current version of the ontologies include modules that provide:

- The formal schema, i.e. classes and properties that can be used to capture application and resource models (TOSCA[12] ontology). This schema has been further enriched for supporting TOSCA Policies, other infrastructure types such as interface types and the association of optimisation DSL with a node template.
- The ontology pattern that should be followed (SODALITE meta-model) in order to define modular and reusable knowledge graphs. Additionally, it contains metadata for the models, needed from the IDE for rendering the view of the models, such as the name of the file, user's information etc.

3.1.3 Location of repository and how to build the code

This component is in the following GitHub repository. The README file of the repository includes the build instructions.

https://github.com/SODALITE-EU/semantic-models

3.1.4 Dependency of component on other components and stand-alone usage of component

The Semantic Knowledge Base is just a database, so it is not dependent on other SODALITE components in order to run. It is dependent on other components in order to populate the database.

3.2 Semantic Reasoner

3.2.1 Description of component

The population of the KB, i.e. the instantiation of the respective ontology patterns to capture resources and applications (AADM), is performed by the Semantic Reasoner, which encapsulates the necessary logic to translate the DSL composed in the IDE by the users to the conceptual model of SODALITE. In addition, the Semantic Reasoner provides a collection of interfaces to retrieve data from the KB for getting information regarding resource and application models, as well as to expose reasoning functionality developed in the IaC layer with respect to searching and validation services (see D3.1, *First version of ontologies and semantic repository*, for more details).

3.2.2 Status of implementation

A number of REST API endpoints have been developed for exposing functionality to external components. Semantic Reasoner mostly interacts with IDE in order to assist users in defining models. Concretely , the component provides:

- The population of the KB with application instances and infrastructure resources.
- Context-assistance services in AADM, RM and optimisation DSL authoring.
- Validation cases for the models.

The access in KB is restricted in APIs by requiring user authentication and authorization.

It should be noted that this REST API exposes additional functionality that has been mainly developed in T4.4 (*Analytics and Semantic Decision Support*) relevant to searching and validation. More details on the backend implementation of the REST API are provided in D4.1 (*IaC Management - initial version*).

3.2.3 Location of repository and how to build the code

This component is in the following GitHub repository. The README file of the repository includes the build instructions.

https://github.com/SODALITE-EU/semantic-reasoner

3.2.4 Dependency of component on other components and stand-alone usage of component

Semantic Reasoner is dependent on the Knowledge Base, and on the Bug Predictor and Fixer for detecting bugs and smells in IaC. Its stand-alone usage is described in the README file of the GitHub repository.

3.3 SODALITE IDE

3.3.1 Description of component

The SODALITE IDE is the visual programming interface between the end users, namely the Application Ops Experts (AOEs) and the Resource Experts (REs), and the SODALITE Infrastructure as Code (IaC) Layer. The IDE also supports Quality Experts to optimise existing models. The IDE enables:

- Application Ops Experts to:
  - Define an Abstract Application Deployment Model (AADM),
  - Select suitable infrastructure/platform resources from the KB that satisfy the requirements of the AADM nodes,
  - Store the AADM into the KB,
  - Triggers the deployment of the AADM within the IAC layer and monitor its deployment status,
  - Browse their AADM stored in the KB, retrieve them into their local IDE workspaces for edition and update, or remove selected ones,
  - Verify the syntactic and semantic correctness of their models,
  - Being assisted by the KB suggestions to complete and improve their models,
  - Visualize graphically their application deployment topologies out of the AADM,
  - Define implementations, as Ansible Models (AMs), for the operations of the interfaces adopted by deployed components.
- Resource Experts to:
  - Model infrastructure/platform resources to be stored into the KB,
  - Map resources and optimisations,
  - Browse their RMs stored in the KB, retrieve them into their local IDE workspaces for edition and update, or remove selected ones,

- Verify the syntactic and semantic correctness of their models,
    - Being assisted by the KB suggestions to complete and improve their models,
    - Store the RM into the KB
- Quality Experts to:
    - Define Optimisation Models (OMs) for target application components. These models will be used by MODAK to improve the components performance at deployment time.
    - Being assisted by the KB suggestions to complete and improve their models,
    - Associate the OMs to application components declared by AOEs in AADMs.

3.3.2 Status of implementation

The development status of the IDE is as follows:

DSL specifications: new versions provide full DSL grammars for AADM, RM, OMs and AMs. AADM and RM are improved, more complete versions of the TOSCA specification for instances (i.e. templates) and types definitions. They include the required modelling elements to fully specify AADMs, RMs, OMs, and AMs for all the use cases.

DSL editors: current version implements the following features:

- Modelling support for designing AADMs: current version supports the modelling of application components and policies, the support of inputs and modules.
- Modelling support for designing RMs: current version supports the modelling of types (data, artifact, capability, interface, relationship, node, policy).
- Modelling support for designing OMs: current version supports the full modeling of the Optimisation DSL designed by SODALITE. See D3.1 (*IaC Management - Intermediate version*).
- Modelling support for designing AMs: current version supports the full modeling of the Ansible DSL designed by SODALITE. See D4.2 (*IaC Management - Intermediate version*).
- Textual modeling of AADMs, RMs, OMs, AMs, with context-aware content assistance, syntax highlighting, outlines, syntactic and semantic validation (not yet supported for AMs).
- Automatic graphical representation of AADMs.
- Complete model governance of AADMs and RMs using the KB as shared repository: browsing of models, retrieval, storage and deletion.
- Deployment of AADMs using the IaC layer.
- Integration with SODALITE IAM Authentication API for secure communications between the IDE and the SODALITE backend components, including the KB

3.3.3 Location of repository and how to build the code

This component is in the following GitHub repository: https://github.com/SODALITE-EU/ide.

The README file of the repository includes instructions to build and install this IDE within an Eclipse instance.

3.3.4 Dependency of component on other components and stand-alone usage of component

The IDE requires the following SODALITE components to properly work:

- The SODALITE Modelling Layer (Knowledge Base)
- The SODALITE IaC layer
- The SODALITE Runtime Layer (Orchestrator)
- The SODALITE Security Support

The SODALITE IDE, based on Eclipse, is a standalone component that requires a local installation, and further configuration to be bound to a target SODALITE backend.

4 IaC Management Components

This section lists the components of the IaC Management layer, describing each component's function, implementation status, and code repository. This section updates the component status reported in D6.5 (*SODALITE Framework - First Version*). A concise list of improvements of the current release over the previous release appears in section 2.1. Additional details on the various features can be found in D4.2 (*IaC Management - Intermediate version*).

4.1 Abstract Model Parser

4.1.1 Description of component

This is a subcomponent of the IaC Blueprint Builder. It parses the JSON representation of the abstract application deployment model (AADM) retrieved from the Semantic Reasoner and prepares an internal representation of topology tree based on the model retrieved. See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version)* and D4.2 (*IaC Management - Intermediate version*) for more information.

4.1.2 Status of implementation

This component has been refactored, improved to cover newly introduced SODALITE features such as using application optimisation model and integrated into the SODALITE's deployment pipeline.

4.1.3 Location of repository and how to build the code

The component build process, Docker image definitions and source code has been implemented inside the following GitHub repository: https://github.com/SODALITE-EU/iac-blueprint-builder.

4.1.4 Dependency of component on other components and stand-alone usage of component

This component depends on the inputs of Semantic Reasoner API and prepares the internal tree model consumed by IaC Blueprint Builder.

4.2 IaC Blueprint Builder

4.2.1 Description of component

This component generates a TOSCA/Ansible[13] blueprint based on the inner abstract tree model provided by the Abstract Model Parser. It has been improved to produce a CSAR containing a valid TOSCA blueprint using execution scripts and artifacts defined in the AADM. After generating the TOSCA blueprint it calls the Orchestrator REST API endpoint to register the blueprint, thus making it ready for deployment.

See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version*) and D4.2 (*IaC Management - Intermediate version*) for more information.

4.2.2 Status of implementation

The IaC Blueprint Builder component has been refactored. Moreover, it has been improved to implement calls to Application Optimiser REST API (MODAK). This way, it is possible to associate HPC TOSCA node templates with optimised image definitions and with job scripts specific to the target execution platform.

4.2.3 Location of repository and how to build the code

The upgraded version of the component is implemented and described inside the following GitHub repository: https://github.com/SODALITE-EU/iac-blueprint-builder.

4.2.4 Dependency of component on other components and stand-alone usage of component

This component depends on the abstract tree model provided by the Abstract Model Parser component and Orchestrator REST API (xopera-rest-api) for registering the created TOSCA CSAR containing the application deployment blueprint.

4.3 Runtime Image Builder

4.3.1 Description of component

This component encapsulates the process of building a SODALITE runtime image based on the input API call data provided by the user (Application Ops Expert) - sample inputs defining sources (e.g. source Dockerfile) and targets (e.g. target image name and tags) can be found under GitHub repository:
https://github.com/SODALITE-EU/image-builder/tree/master/REST_API/JSON-build-params.
After running the image-builder workflow, the created image is pushed to the Image Registry to be later pulled and deployed by the Orchestrator at deployment time.

See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version)* and D4.2 (*IaC Management - Intermediate version*) for more information.

4.3.2 Status of implementation

The improved version of the component that comprises the IaC TOSCA/Ansible definitions of sources and targets and a REST API endpoint for invoking the Image Builder has been implemented. Additionally a CLI Docker image has been provided for user convenience.

4.3.3 Location of repository and how to build the code

The improved version of the component is implemented and described inside this GitHub repository: https://github.com/SODALITE-EU/image-builder.

4.3.4 Dependency of component on other components and stand-alone usage of component

This component depends on the inputs of the AOE from the IDE component to support the process of building a specific image. The Image Builder can be used as a standalone component using CLI Docker image implementation or as a REST API.

4.4 Concrete Image Builder

4.4.1 Description of component

This is a subcomponent that generates a specific implementation of a runtime image (Docker - for instance). The built image is then pushed to the Docker Image Registry to be later pulled and deployed by the Orchestrator at deploy time. The creation of the concrete image depends on the input API call data provided by the user (Application Ops Expert). Sample inputs defining sources and targets can be found under GitHub repository:
https://github.com/SODALITE-EU/image-builder/tree/master/REST_API/JSON-build-params.

See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version)* and D4.2 (*IaC Management - Intermediate version*) for more information.

4.4.2 Status of implementation

The intermediate version of the component that comprises the IaC TOSCA/Ansible definitions of sources and targets and a REST API endpoint for invoking the Image Builder was implemented for building Docker images and pushing them to the SODALITE Image Registry.

4.4.3 Location of repository and how to build the code

The initial version of the component is implemented inside this GitHub repository: https://github.com/SODALITE-EU/image-builder.

4.4.4 Dependency of component on other components and stand-alone usage of component

This component depends on the internal image creation model created by the Runtime Image Builder and Image Registry API for pushing specific container image types. This is a sub component, so it cannot be run without Runtime Image Builder.

4.5 Image Registry

4.5.1 Description of component

This component implements a private SODALITE Image Registry. The images built within SODALITE are stored in this private Image Registry and later pulled and deployed by the Orchestrator at deployment time.

See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version)* and D4.2 (*IaC Management - Intermediate version*) for more information.

4.5.2 Status of implementation

The intermediate version of the private Docker registry has been built, configured and secured with TLS using IaC TOSCA/Ansible blueprint, which was deployed using SODALITE's Orchestrator xOpera.

4.5.3 Location of repository and how to build the code

The intermediate version of the component is implemented as IaC inside this GitHub repository: https://github.com/SODALITE-EU/iac-platform-stack. This repository also comprises examples and deployment blueprints for SODALITE use cases deployed through SODALITE platform.

4.5.4 Dependency of component on other components and stand-alone usage of component

The Image Registry can be and is used as a standalone component.

4.6 Application Optimiser

4.6.1 Description of component

This component, named MODAK, tries to build a performance optimised runtime given the target platform and configuration and predefined optimisation options.

See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version*), D3.3 (*Prototype of application and infrastructure performance models - First version*) and D4.2 (*IaC Management - Intermediate version*) for more information.

4.6.2 Status of implementation

The initial version of performance model training is released for building a performance model. The MODAK code is available in the public repository. Currently, MODAK supports TensorFlow[14], PyTorch[15], MXnet[16], mpich[17], and openmpi[18] containers for x86 and NVIDIA GPUs.

See D4.2 (*IaC Management - Intermediate version*) and D6.3 (*Intermediate Implementation and Evaluation of the SODALITE Platform and Use Cases*) for more information.

4.6.3 Location of repository and how to build the code

The initial version of the component is implemented and described inside this GitHub repository: https://github.com/SODALITE-EU/application-optimisation.

4.6.4 Dependency of component on other components and stand-alone usage of component

Initial integration of MODAK in the SODALITE framework has been implemented (see D4.2). MODAK itself can be used as a standalone package, following the instructions available in the component repository. Integration-wise MODAK has been used in IaC Blueprint Builder calls for retrieving the optimised image and job script for the target platform.

4.7 IaC Verifier

4.7.1 Description of component

This component provides a unified REST API for the verification capabilities of Topology Verifier and Provisioning Workflow Verifier.

See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version)* and D4.2 (*IaC Management - Intermediate version*) for more information.

4.7.2 Status of implementation

The REST API was implemented to expose all key verification capabilities of Topology Verifier and Provisioning Workflow Verifier.

4.7.3 Location of repository and how to build the code

This component is a sub-project under the following GitHub repository. The README file of the project includes the build instructions.

https://github.com/SODALITE-EU/verification

4.7.4 Dependency of component on other components and stand-alone usage of component

This component exposes the capabilities provided by Topology Verifier and Provisioning Workflow Verifier as RESTful APIs, and depends on those components.

4.8 Verification Model Builder

4.8.1 Description of component

This component builds the models (e.g., semantic models and Perti-net based models) required for the verification of a given set of IaC artifacts (TOSCA and Ansible).

See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version)* and D4.2 (*IaC Management - Intermediate version*) for more information.

4.8.2 Status of implementation

The current version of this component can build models required for checking TOSCA files against the constraints defined by the TOSCA standard. The PetriNet models can be built from the description files in Petri Net Markup Language (PNML).

4.8.3 Location of repository and how to build the code

This component is a sub-project under the following GitHub repository. The README file of the project includes the build instructions.
https://github.com/SODALITE-EU/verification

4.8.4 Dependency of component on other components and stand-alone usage of component

There are no dependencies, and can be used as a standalone component.

4.9 Topology Verifier

4.9.1 Description of component

This component verifies the constraints over the structures of the TOSCA blueprints. Topology Verifier uses the verification capabilities provided by OpenStack TOSCA Parser[19] to detect the violations of the syntax of blueprints with respect to the TOSCA specification.

See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version*) and D4.2 (*IaC Management - Intermediate version*) for more information.

4.9.2 Status of implementation

The verification of TOSCA models for syntax errors and non-compliance to the standard (TOSCA Simple Profile in YAML Version 1.2) has been implemented.

4.9.3 Location of repository and how to build the code

This component is a sub-project under the following GitHub repository. The README file of the project includes the build instructions.

https://github.com/SODALITE-EU/verification

4.9.4 Dependency of component on other components and stand-alone usage of component

This component uses Verification Model Builder , and can be used as a standalone component.

4.10 Provisioning Workflow Verifier

4.10.1 Description of component

This component verifies the constraints over the deployment workflow of the application described in the Ansible (IaC) scripts. We employ Petri net as the formal modelling language, which is widely used for verifying workflows and business processes.

See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version)* and D4.2 (*IaC Management - Intermediate version*) for more information.

4.10.2 Status of implementation

The initial support for translating the control flow model of the Ansible plays to Petri Net Markup Language (PNML) has been implemented.

4.10.3 Location of repository and how to build the code

This component is a sub-project under the following GitHub repository. The README file of the project includes the build instructions.

https://github.com/SODALITE-EU/verification

4.10.4 Dependency of component on other components and stand-alone usage of component

This component uses Verification Model Builder , and can be used as a standalone component.

4.11 Bug Predictor and Fixer

4.11.1 Description of component

This component predicts different types of bugs in TOSCA and Ansible. This includes the taxonomies covering a wide-range of best/bad practices, bugs and software smells for IaC such as code smells, design smells, security smells, and linguistic anti-patterns. Then, the tools support for detecting and correcting those bugs are also developed.

See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version*) and D4.2 (*IaC Management - Intermediate version*) for more information.

4.11.2 Status of implementation

We developed three types of taxonomies for IaC: best/bad practices taxonomy, smell taxonomy, and bug taxonomy. We have developed the support for detecting TOSCA smells with semantic reasoning, Ansible smells with rule-based reasoning, and Ansible linguistic anti-pattern with NLP and deep learning.

4.11.3 Location of repository and how to build the code

This component is a sub-project under the following GitHub repository: https://github.com/SODALITE-EU/defect-prediction. The README file of the project includes the build instructions.

4.11.4 Dependency of component on other components and stand-alone usage of component

This component uses Predictive Model Builder and IaC Quality Assessor components.

4.12 Predictive Model Builder

4.12.1 Description of component

This component builds the models that can find the smells in TOSCA and Ansible artifacts. A model can be a rule-based model, a heuristics based model, and a data-driven (deep learning) model.

See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version*) and D4.1 (*IaC Management - initial version*) for more information.

4.12.2 Status of implementation

A semantic rule based model was built to detect the security smells in TOSCA. An informal rule based model (Ansible-Lint Rules) was built to detect the code and security smells in Ansible. The deep learning and NLP based models were built to detect linguistic anti-patterns in Ansible.

4.12.3 Location of repository and how to build the code

This component is a sub-project under the following GitHub repository: https://github.com/SODALITE-EU/defect-prediction. The README file of the project includes the build instructions.

4.12.4 Dependency of component on other components and stand-alone usage of component

This component is used by Bug Predictor and Fixer, and depends on Semantic Reasoner and KB.

4.13 IaC Quality Assessor

4.13.1 Description of component

This component can calculate different software quality metrics for IaC artifacts.

See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version*) and D4.1 (*IaC Management - initial version*) for more information.

4.13.2 Status of implementation

A set of the metrics for Ansible was developed. A REST API was also developed.

4.13.3 Location of repository and how to build the code

This component is at the following GitHub repository: https://github.com/SODALITE-EU/iac-quality-framework. The README file of the repository includes the build instructions.

4.13.4 Dependency of component on other components and stand-alone usage of component

This component is used by Bug Predictor and Fixer. This component can be used as a standalone component/library.

4.14 Platform Discovery Service

4.14.1 Description of component

The Platform Discovery Service creates a TOSCA platform resource model definition based on access data and type provided by the Resource Expert. See D2.2 (*Requirements, KPIs, evaluation* *plan and architecture - Intermediate version*) and D4.2 (*IaC Management - Intermediate version*) for more information.

4.14.2 Status of implementation

Platform Discovery Service exposes an API able to provide TOSCA resource model definitions for HPC clusters like Torque/Slurm, Openstack private Cloud and partially AWS.

4.14.3 Location of repository and how to build the code

This component is described and implemented under the following GitHub repository: https://github.com/SODALITE-EU/platform-discovery-service.

4.14.4 Dependency of component on other components and stand-alone usage of component

This component depends on the inputs from the IDE component provided by the Resource Expert as platform project definition and platform access data (credentials) for accessing and running discovery of the platform.

4.15 IaC Model Repository

4.15.1 Description of component

IaC Model Repository is a part of the Knowledge Base and contains:

- Performance Model of an infrastructure based on benchmarks.
- Performance Model of an Application based on scaling runs done in the past.
- Mapping of optimisations and applications and their suitability for a particular infrastructure.
- Optimisation recipe for a particular deployment. This contains selected optimisations by the user for an application and infrastructure target.

4.15.2 Status of implementation

A preliminary version of the repository is implemented as a MySQL database. A Python script is used to access the stored data. It will extend in Y3 to have full access from the MODAK performance optimiser.

4.15.3 Location of repository and how to build the code

The current implementation of the component is described and implemented under the following GitHub repository: https://github.com/SODALITE-EU/application-optimisation.

4.15.4 Dependency of component on other components and stand-alone usage of component

IaC Model Repository interacts with the SODALITE IDE and contains the Performance Model of infrastructure and application (offline analysis). This component can be used as a standalone component.

5 Runtime Layer Components

This section lists the components of the Runtime Layer, describing each component's function, implementation status, and code repository. This section updates the component status reported in D6.5 (*SODALITE Framework - First Version*). A concise list of improvements of the current release over the previous release appears in section 2.1. Additional details can be found in deliverable D5.2 (*Application deployment and dynamic runtime - intermediate version*).

5.1 Orchestrator - xOpera

5.1.1 Description of component

SODALITE uses xOpera as the base orchestrator. xOpera is an open source lightweight TOSCA 1.3 Simple YAML compliant orchestrator, which uses Ansible as the actuation scripting language for agentless provisioning, configuration and application lifecycle management on heterogeneous platforms.

5.1.2 Status of implementation

xOpera aims to be a simple and lightweight TOSCA 1.3 compliant orchestrator. SODALITE developed and deployed a dockerized version of the REST API implementation of xOpera functionalities to the testbed by a prepared IaC deployment blueprint. The development of xOpera is ongoing as it aims to implement TOSCA yaml 1.3 standard. Currently used version of xOpera is 0.6.4.

5.1.3 Location of repository and how to build the code

This component is referenced as a GitHub submodule in the following GitHub repository: https://github.com/SODALITE-EU/orchestrator.

5.1.4 Dependency of component on other components and stand-alone usage of component

xOpera is a stand-alone tool, independent of other SODALITE components.

5.2 xOpera REST API

5.2.1 Description of component

xOpera REST API implements endpoint interfaces encapsulating xOpera orchestrator functionalities and extending deployments with blueprint persistence, session management, status of deployment, history of deployment documented with Swagger UI. The dockerized REST API enables transportability and easy integration with other SODALITE components.

5.2.2 Status of implementation

xOpera REST API is developed inside the SODALITE project and currently supports deployments to platforms such as HPC (Torque/Slurm), Openstack, AWS and Kubernetes. Recent extensions to xOpera and the REST API support the targeted functionalities needed in SODALITE with special focus on concurrent multi-platform application deployment, resuming deployments and deployment reconfiguration.

5.2.3 Location of repository and how to build the code

This component is implemented and developed in the following GitHub repository: https://github.com/SODALITE-EU/xopera-rest-api. The README files provide extensive information on setting up a working REST API environment.

5.2.4 Dependency of component on other components and stand-alone usage of component

xOpera REST API is a stand-alone tool, independent of other SODALITE components.

5.3 Deployment Refactorer

5.3.1 Description of component

The refactoring of the deployment model of a running application is performed in response to the potential violations of the application goals, which is determined using the monitoring data. The refactoring can find and enact a new deployment model for the application that can resolve the detected goal violations. The refactoring decisions making employs both rule-based techniques as well as machine learning based techniques. Refactorer can also detect the anomalies in the performance of the application deployment. See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version*) and D5.2 (*Application deployment and dynamic runtime - Intermediate version*) for more information.

5.3.2 Status of implementation

We developed the component and demonstrated its practicality and feasibility by applying it to 1) an extension of the RuBiS benchmark application[20] [21] deployed on Google's Compute Engine (92 deployment alternatives), 2) Teastore microservice benchmark[22] application on Google Kubernetes Engine (78 deployment alternatives) and SODALITE snow use case. The experimental results using the predictive algorithms demonstrated the effectiveness of our proposed approach, accurately predicting the performance of deployment configurable cloud applications. We developed the rule-based adaptation support, and validated it with three key scenarios in the Vehicle IoT use case: location-aware redeployment, alert-driven redeployment with Cloud alerts, alert-driven redeployment with Edge alerts. We have been experimenting with a recent microserice memory anomaly dataset[23] using three predictive algorithms Random Forest, Decision Tree[24] and Deep Learning with AdaBoost[25]. We have started to collect a service network anomaly dataset with TeaStore microservice benchmark[26], Google kubernetes Engine, and SODALITE SkyDive monitoring support.

5.3.3 Location of repository and how to build the code

This component is in the following GitHub repository. The README file of the repository includes the build instructions.

https://github.com/SODALITE-EU/refactoring-ml

5.3.4 Dependency of component on other components and stand-alone usage of component

This component depends on Monitoring Layer, Orchestrator, Node Manager, Refactoring Option Discoverer, and Deployment Preparation API.

5.4 Node Manager

5.4.1 Description of component

Node Manager is in charge of managing existing resources deployed by the SODALITE users or by other SODALITE components. Node Manager goal is to fulfil requirements on the response time while minimizing resource consumption of multiple concurrent applications. Node Manager operates on three levels: at the cluster level, it gathers the applications' requests that are efficiently scheduled using custom algorithms on fast GPUs or CPUs. At the machine level, it manages resource contention scenarios that arise from the execution of multiple concurrent containers on the machine. At the container level, control-theoretical planners continuously re-configure containers' CPU resources (vertical scalability) given the current workload, performance and the utilization of the GPUs.

5.4.2 Status of implementation

Node Manager is implemented in Python as a set of distributed components. It exploits Kubernetes to ease the deployment of containerized applications. The entrypoint of the Node Manager is a launcher that takes a TOSCA blueprint with minimal information about the user applications and automatically generates Kubernetes deployments and starts the control loops. Node Manager was tested with four benchmark applications: Skyline Extraction from SnowUC, GoogLeNet, ResNet, and VGG-16. Additional details can be found in deliverable D5.2 (*Application deployment and dynamic runtime - intermediate version*).

5.4.3 Location of repository and how to build the code

This component is in the following GitHub repository. The README file in the repository contains architecture details and run instructions.

https://github.com/SODALITE-EU/refactoring-ct

5.4.4 Dependency of component on other components and stand-alone usage of component

This component currently does not depend on any SODALITE component. In Y3, it will be integrated with the Monitoring and Deployment Refactorer.

5.5 Refactoring Option Discoverer

5.5.1 Description of component

Refactoring Option Discoverer supports discovering TOSCA Compliant deployment options by leveraging semantic matchmaking. The search criteria can be user-defined constraints, infrastructure design patterns and anti-patterns. See D2.2 (*Requirements, KPIs, evaluation plan and architecture - Intermediate version*) and D5.2 (*Application deployment and dynamic runtime - Intermediate version*) for more information.

5.5.2 Status of implementation

Refactoring Option Discoverer supports matchmaking based on node properties, node capabilities and requirements, and node policies (TOSCA policies).

5.5.3 Location of repository and how to build the code

This component is in the following GitHub repository. The README file of the repository includes the build instructions.

https://github.com/SODALITE-EU/refactoring-option-discoverer

5.5.4 Dependency of component on other components and stand-alone usage of component

This component depends on the Semantic Reasoner.

5.6 ALDE

5.6.1 Description of component

ALDE[7] (Application Lifecycle Deployment Engine) is a REST API responsible for managing the workload scheduling and execution of applications, primarily intended for HPC environments.

5.6.2 Status of implementation

Since it is an outcome of the TANGO[27] project, an extension of the support to the Torque scheduler for HPC environments has been developed. The development has been completed and tested together with the rest of the innovations such as the CI/CD integration to the DockerHub.

5.6.3 Location of repository and how to build the code

This component is in the following GitHub repository:
https://github.com/SODALITE-EU/alde
The code and installation manual are located inside the repository. The README file in the repository contains the installation description together with some configuration examples.

5.6.4 Dependency of component on other components and stand-alone usage of component

ALDE requires an HPC environment in order to be operative, therefore an initial setup is necessary to configure the communications and the administrative permissions on the HPC side. Once a user is created and placed in the ALDE server, ALDE is fully capable of operating the HPC environment deploying Singularity containers by using its REST API interface.

5.7 Monitoring - Prometheus/Consul

5.7.1 Description of component

Prometheus[28] is an open source system monitoring and alerting toolkit. Exporters deployed on the various machines within a cluster periodically collect system information, which is stored in a central database. The data can then be queried, filtered and analyzed according to the needs of the client (either an end-user or another component of the SODALITE stack). Prometheus is not responsible for keeping a list of monitoring targets, this task being delegated to the Consul instance. Consul offers a REST API through which services can be registered, the properties of each service being specified in a JSON file passed along with the registration request. Prometheus can be configured to retrieve exporter endpoints seamlessly from the Consul service registry.

5.7.2 Status of implementation

Prometheus and Consul are baseline, open source, third party components adopted by SODALITE. See dynamic monitoring in D5.2 (*Application deployment and dynamic runtime - intermediate version*). Additionally, TOSCA based blueprints and Ansible playbooks have been implemented for supporting Prometheus deployment as part of the SODALITE stack.

5.7.3 Location of repository and how to build the code

Prometheus source code may be cloned from https://github.com/prometheus/prometheus. Instructions to configure and run Prometheus are in the following location: https://prometheus.io/docs/prometheus/latest/getting_started/. Consul can be download from its Web site: https://www.consul.io/

See D5.2 for additional details on the adoption of Prometheus in the SODALITE stack.

5.7.4 Dependency of component on other components and stand-alone usage of component

In SODALITE, Prometheus requires integration with Consul and the registered exporters.

5.8 Alert Manager/Rule File Server

5.8.1 Description of component

AlertManager handles the alerts triggered by the Prometheus server and notifies any component or actor subscribed to them, such as the Orchestrator and the Refactoring, which apply adaptations in the resources available according to any relevant change on the infrastructure derived from the alerts triggered. These alerts are defined for each application on rule files that are kept by the Rule File Server. This server offers a REST API that allows the applications to register rule files in order to be evaluated by the Prometheus component, which is properly configured to access those files.

5.8.2 Status of implementation

The Alert Manager is deployed in its own container using the official Docker image and registered within Prometheus. An implementation of the Rule Server, written in Python, with Flask and Gunicorn is deployed as a Docker container and offers the Rule File API to register/unregister alerting rules.

5.8.3 Location of repository and how to build the code

The repository (https://github.com/SODALITE-EU/monitoring-system/tree/master/ruleserver) is hosted inside of the SODALITE GitHub page and it contains the build procedure and a description of its functioning.

5.8.4 Dependency of component on other components and stand-alone usage of component

Alert Manager requires a Prometheus instance for registration.

5.9 Node Exporter

5.9.1 Description of component

Node Exporter[29] is a piece of software that exposes hardware metrics to be gathered by Prometheus. It gathers metrics related to CPU (such as usage, frequency, etc), memory, I/O and network, among others, providing an insight on resource utilization.

5.9.2 Status of implementation

Node Exporter is a baseline, open source, third party exporter shipped within Prometheus and adopted by SODALITE. At this reporting stage, Node Exporter automatic deployment has been improved to support Consul registration (See Node Exporter in D5.2).

5.9.3 Location of repository and how to build the code

A list of the metrics that Node Exporter exposes can be found in its repository together with its requirements and installation process. (https://github.com/prometheus/node_exporter).

5.9.4 Dependency of component on other components and stand-alone usage of component

Node Exporter requires a Consul instance for registration.

5.10 IPMI Exporter

5.10.1 Description of component

IPMI Exporter[30] is another Prometheus exporter created to expose the power measurements of the physical nodes given by the output of one command. It can also be adapted to expose any metric that is outputted by any CLI command.

5.10.2 Status of implementation

IPMI Exporter is a SODALITE open source component and can accept modifications performed or suggested by the community. In its current state, it only exposes one metric, but this can be easily expanded if more are required. IPMI Exporter automatic deployment has been improved to support Consul registration (See IPMI Exporter in D5.2).

5.10.3 Location of repository and how to build the code

The repository (https://github.com/SODALITE-EU/ipmi-exporter) is hosted inside of the SODALITE GitHub page and it contains the build procedure and a description of its functioning.

5.10.4 Dependency of component on other components and stand-alone usage of component

IPMI Exporter requires a Consul instance for registration.

5.11 HPC Exporter

5.11.1 Description of component

The HPC Exporter collects metrics about different aspects of HPC infrastructure usage from both performance and cost viewpoints: job status, CPU and wall time consumption, physical and virtual memory consumption, energy consumption, traffic for I/O and network communications, etc. Different types of HPC job schedulers are supported, starting with PBS Professional and Slurm in the current implementation.

5.11.2 Status of implementation

HPC Exporter implementation is completed as a standalone exporter, packaged within a Docker container, and configurable through the command line interface. Upon initialization, it registers itself in Consul so that it gets required by Prometheus to provide metrics about job execution in configured target HPC clusters. See HPC Exporter documentation in GitHub repository link.

5.11.3 Location of repository and how to build the code

The repository (https://github.com/SODALITE-EU/hpc-exporter) is hosted inside of the SODALITE GitHub page and it contains the build procedure and a description of its functioning.

5.11.4 Dependency of component on other components and stand-alone usage of component

HPC Exporter requires a Consul instance for registration.

5.12 Skydive

5.12.1 Description of component

Skydive[31] is a real-time network topology and protocol analyzer that provides detailed network topology and performance information. Skydive agents collect topology information and flows and forward them to a central agent for further analysis. A more detailed summary is in deliverable D5.1 (*Application deployment and dynamic runtime - initial version*) and additional details are available on the Skydive site (http://skydive.network/documentation/).

5.12.2 Status of implementation

Skydive is a stable open source project with community support. New features are added on an on-going basis. IBM team members are regular contributors to the Skydive community. In 2019, IBM team members implemented and released a generic flow exporter for Skydive to expose the Skydive flow data in a convenient form to various consumers. The prometheus-skydive connector is built on top of the flow exporter. In 2020, the prometheus-skydive connector was hardened and was pushed upstream. A blog describing the use of the prometheus-skydive connector was prepared and published: http://skydive.network/blog/prometheus-connector.html.

5.12.3 Location of repository and how to build the code

The Skydive code repository is at https://github.com/skydive-project/skydive. Details on how to deploy Skydive are at http://skydive.network/documentation/getting-started. Pre-compiled versions of Skydive are availabe at https://github.com/skydive-project/skydive/releases. Details on how to compile the code from source are at http://skydive.network/documentation/build.

The Skydive flow exporter code repository is at
https://github.com/skydive-project/skydive-flow-exporter, including instructions on how to build the code. Code and instructions to compile and use the prometheus-skydive connector are at
https://github.com/skydive-project/skydive-flow-exporter/tree/master/prom_sky_con.
A documentation on using the prometheus-skydive connector is at
http://skydive.network/blog/prometheus-connector.html.
A documentation on the general use of the Skydive flow exporter is at
http://skydive.network/blog/exporters.html.
Adjustments of the prometheus-skydive connector to be used with SODALITE can be found in
https://github.com/SODALITE-EU/skydive-flow-exporter/tree/master/prom_sky_con. These adjustments include creating a Docker container of the connector, and placing the container in the SODALITE repository upon successful build using Jenkins.

5.12.4 Dependency of component on other components and stand-alone usage of component

The Skydive tool is comprised of agents that run on each of the monitored machines plus an analyzer that collects all the data from the various agents. Native host-based tools are used to collect the topology and metrics information. The core Skydive flow exporter depends on data supplied by the Skydive analyzer, but is otherwise independent of other tools. The prometheus-skydive connector is dependent on both the Skydive flow exporter and a Prometheus server. Data from flows that are captured using Skydive is packaged and forwarded to Prometheus. Beyond this specific function of acting as a connector, the prometheus-skydive connector is independent of other SODALITE components.

5.13 Edge Exporter

5.13.1 Description of component

The Edge Exporter is a meta package that includes multiple exporters for exposing metrics of specific heterogeneous accelerators found in Edge nodes. The accelerator metrics exposed are primarily the number of devices, utilization rates, and temperature measurements (where supported), in order to facilitate thermal-based alerting and reconfiguration.

5.13.2 Status of implementation

At present, initial implementations for the EdgeTPU and Intel Neural Compute Stick 2 (NCS2) have been completed. Thermal properties of integrated GPUs are already captured and exported via the aforementioned Node Exporter, so no GPU-specific exporter is presently required. As the integrated GPUs are exposed in different thermal zones on different Edge nodes, customized alerting rules must be provided on a per-node basis (this is expected to be simplified through the utilization of the alerting rule server in Y3).

The NCS2 exporter presently exposes device counts by default, and requires manual instrumentation in any application using the device in order to make temperature readings available. This is due to a limitation of the OpenVINO[32] runtime, which requires a machine learning model to be loaded onto the device before the temperature reading is exposed internally. A further limitation is that any use of the OpenVINO runtime directly by the exporter prohibits the device from being used by other applications, so this limitation is unlikely to go away in the near future. The intended deployment scenario and application instrumentation can be seen in the figure below:

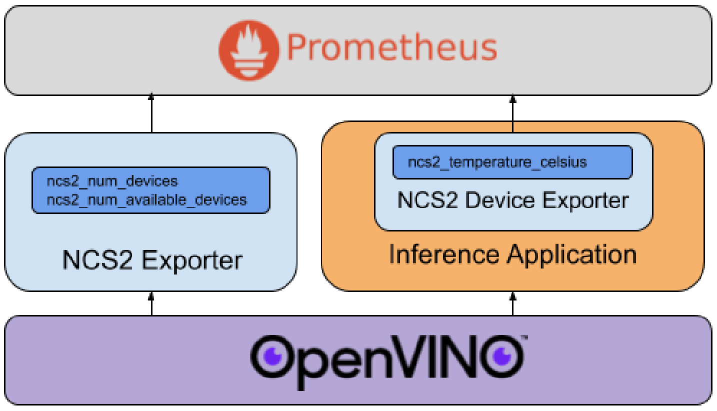

Figure 5 - NCS2 Exporter instrumentation requirements

5.13.3 Location of repository and how to build the code

The EdgeTPU exporter, together with building, usage, and deployment instructions is available at: https://github.com/adaptant-labs/edgetpu-exporter

The NCS2 exporter, together with building, usage, and deployment instructions is available at: https://github.com/adaptant-labs/prometheus_ncs2_exporter

5.13.4 Dependency of component on other components and stand-alone usage of component

Each exporter is provided as a Docker container which can be run standalone, or as part of a managed Kubernetes deployment. Kubernetes manifests are provided with each exporter to support node-local and cluster-wide scheduling, and external Helm charts are also available. When deployed in Kubernetes, each exporter Pod is annotated with a Prometheus scraping annotation, which ensures that each exporter is automatically detected and scraped by the Prometheus server without any additional configuration required.

6 Conclusion

This document summarizes the SODALITE Framework at the end of the second year of the project. This comprises what has been achieved for the second SODALITE prototype embodied in Milestone 6 (MS6) of the project. This includes components from all three layers of the Sodalite architecture: the Modelling Layer, the Infrastructure as Code Layer, and the Runtime Layer.

Improvements of many parts of the prototype are included in the updated SODALITE platform.

- Automated discovery and TOSCA description of infrastructure.
- Support for the creation of Ansible scripts integrated with the Resource Models.
- Static optimisation of applications before deployment (MODAK).
- Semantic and Analysis Support including: Bug Taxonomy; Unified best and bad practices Catalog; Unified Smell Catalog; Linguistic Anti-pattern detection via NLP and Deep Learning.
- Orchestration: Improvements to xOpera including Reconfiguration; Parallel execution of deployment; Resume command; Worklows; REST API.
- Dynamic Monitoring.
- Refactoring.
- Modelling: Support of TOSCA Policies in the Modelling Layer; Improved context assistance in model authoring; Improved scalability in the semantic services.

All SODALITE components are built and deployed using GitHub and Jenkins.

Sodalite Use-Cases all run on the prototype. Additional details can be found in D6.3 (*Intermediate Implementation and Evaluation of the SODALITE Platform and Use Cases*).

Instructions are provided individually for each component on how to compile and use the component. There is also a blueprint from which it is possible to deploy the entire SODALITE stack.

Video demonstrations of many of these components are presented on the Project's YouTube channel.

The next update of this document is D6.7 *SODALITE framework - Final version*, scheduled for M36 of the project.

7 References

1. https://hub.docker.com/u/sodaliteh2020
2. https://github.com/SODALITE-EU/iac-platform-stack/tree/master/docker-local
3. https://www.sodalite.eu/reports/d24-guidelines-contributors-sodalite-framework
4. https://www.youtube.com/channel/UCrArVp55GaJs78jFt1lUfFg
5. https://www.sodalite.eu/reports/d21-requirements-kpis-evaluation-plan-and-architecture-first-version
6. https://www.sodalite.eu/reports/d31-first-version-ontologies-and-semantic-repository
7. https://github.com/SODALITE-EU/iac-platform-stack
8. https://www.keycloak.org/ Open Source Identity and Access Management for Modern Applications and Services
9. https://www.vaultproject.io/ Secure, store and tightly control access to tokens, passwords, certificates, encryption keys for protecting secrets and other sensitive data using a UI, CLI, or HTTP API.
10. https://docs.google.com/document/d/1w6wYJbTZvBbt5LD6sXReXbx1uPDjefYFAU5KEv8X_8w
11. http://graphdb.ontotext.com/
12. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
13. https://www.ansible.com/
14. TensorFlow: an end-to-end open source platform for machine learning, https://www.tensorflow.org/, 2021.
15. PyTorch: an open source machine learning framework, https://pytorch.org/, 2021.
16. MXnet: a deep learning framework designed for both efficiency and flexibility, https://mxnet.apache.org/, 2021.
17. MPICH: High performance and portable MPI, https://www.mpich.org/, 2021.
18. OpenMPI: Open Source High Performance Computing, https://www.open-mpi.org/, 2021.
19. https://github.com/openstack/tosca-parser
20. https://github.com/uillianluiz/RUBiS
21. https://github.com/SODALITE-EU/refactoring-ml/tree/master/benchmarks-apps
22. https://github.com/DescartesResearch/TeaStore
23. Lomio, Francesco, et al. "RARE: a labeled dataset for cloud-native memory anomalies." Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation. 2020.
24. Ali, Jehad, et al. "Random forests and decision trees." International Journal of Computer Science Issues (IJCSI) 9.5 (2012): 272.
25. Schapire, Robert E. "Explaining adaboost." Empirical inference. Springer, Berlin, Heidelberg, 2013. 37-52.
26. Eismann, Simon, et al. "TeaStore: A Micro-Service Reference Application for Cloud Researchers." 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). IEEE, 2018.
27. http://www.tango-project.eu/content/application-lifecycle-deployment-engine-alde
28. https://prometheus.io
29. https://github.com/prometheus/node_exporter
30. https://github.com/SODALITE-EU/ipmi-exporter
31. http://skydive.network
32. https://docs.openvinotoolkit.org/latest/index.html
33. Prometheus is an open source baseline component
34. Consul is a open source baseline component
35. Prometheus is a opensource baseline component
36. https://github.com/uillianluiz/RUBiS
37. https://github.com/SODALITE-EU/refactoring-ml/tree/master/benchmarks-apps
38. Lomio, Francesco, et al. "RARE: a labeled dataset for cloud-native memory anomalies." Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation. 2020.
39. Eismann, Simon, et al. "TeaStore: A Micro-Service Reference Application for Cloud Researchers." 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). IEEE, 2018.

Appendix A - Reference Component Definition

In this appendix, we collect all the component status information and present it in a compact, tabular format.

Appendix A.1 SODALITE Semantic Modelling Layer

A.1.1 SODALITE IDE

Module name	IDE
GitHub location	https://github.com/SODALITE-EU/ide
Documentation	README file
Downloadable artifacts	IDE Docker image
Development status	Released versions: 1. M24 Release - 31/01/2021 a. Model (AADM, RM) Governance i. CRUD Operations on KB ii. Governance view b. Improved support of policies in AADM and RM c. Support for modules in AADM and RM d. Improved content assistance for AADMs and RMs e. AAI communications with SODALITE backend f. Improved IDE configuration g. Improved IDE tutorial h. Extensions in DSLs for AADMs and RMs i. Improved AADM visual representation j. Improved AADM deployment wizard k. AADM outline l. Improvements in syntactic and semantic validation m. Ansible DSL and editor n. Improved modeling (AADM, RM) of all SODALITE Use Cases o. IDE update site in GitHub Pages p. Update to latest Eclipse version 2. M18 Release - 02/10/20 a. AADM and RM DSLs b. AADM and RM textual editors c. Content assistance for AADMs and RMs d. Saving AADMs and RMs in KB e. Syntactic and semantic validation using the KB

	i. notification of reported issues in model editor f. AADM deployment using IaC and Runtime Layers g. Extensions in AADM and RM grammars for UC requirements h. Optimisation model DSL and editor i. KB optimisation suggestions i. AADM deployment wizard j. AADM creation wizard k. AADM visual representation l. IDE configuration m. IDE build and publish pipeline for CI/CD i. IDE update site ii. IDE Docker container n. Modeling (AADM, RM) of all UCs o. IDE tutorial
Deployment status	The IDE is a standalone component to be installed locally by the end-user from the IDE Eclipse update site as explained in the README file. Alternatively, an IDE container can be instantiated from the IDE Docker image
Integration status	The IDE interacts with the following SODALITE backend services: ● KB for CRUD operations on models (AADM, RM) stored in the KB, as well as to get KB knowledge inference and semantic validation for model authoring. ● IaC Layer, to register AADMs to be deployed ● Runtime Layer, concretely with the Orchestrator, to deploy AADMs ● AAI Layer, to authenticate the IDE user and get the security token required for securing all IDE-backend interactions.
Dependencies	The IDE depends on the KB, the IaC Builder, the Orchestrator and the KeyCloak components.
Next steps	● AADM Deployment and Lifecycle Governance view ● AADM visual modeling ● Further required improvements in existing features ● Further required integration requirements

Table 2 - Development status of the IDE

A.1.2 Semantic Reasoner

Module name	Semantic Reasoning Engine
GitHub location	https://github.com/SODALITE-EU/semantic-reasoner
Documentation	README file
Downloadable artifacts	Semantic Reasoner Docker image
Development status	Released versions: 1. Current version - M24 Release - 31/01/21 a. User Management b. Support of TOSCA Policies c. Enriched context-assistance 2. 0.2.2 - 10/12/20 d. Support of workspaces 3. M18Release - 02/10/20 e. Optimisation DSL suggestions f. Making DSL more abstract 4. 0.1.0 g. Provides the basic reasoning infrastructure to WP4 for developing, searching and validation services; it provides the REST API that can be used to save and get data from the semantic triple store.
Deployment status	Semantic Reasoning Engine has been deployed in a Docker container on a cloud testbed.
Integration status	Semantic Reasoning Engine is integrated with: ● IDE (It pushes DSL definitions using the REST API through HTTP calls) ● AAI using Auth API ● Bug Predictor (detects security smells on TOSCA)
Dependencies	This module depends on Semantic Knowledge Base and Bug Predictor.
Next steps	Providing more intelligent context-assistance services, and more validation cases.

Table 3 - Development status of Semantic Reasoning Engine

Module name	Semantic Population Engine
GitHub location	https://github.com/SODALITE-EU/semantic-reasoner
Documentation	README
Downloadable artifacts	Semantic Reasoner Docker image
Development status	Released versions: 1. Current version - M24 Release - 31/01/21 a. User Management b. Support of TOSCA Policies 2. 0.2.2- 10/12/20 c. Support of workspaces 3. 0.1.0 d. Saving infrastructure resources e. Saving application resources
Deployment status	Semantic Reasoning Engine has been deployed in a Docker container on a cloud testbed.
Integration status	Semantic Reasoning Engine is integrated with: ● IDE (It pushes DSL definitions using the REST API through HTTP calls) ● AAI using Auth API
Dependencies	This module depends on Semantic Knowledge Base and Bug Predictor.
Next steps	Covering more coverage to the TOSCA Specification

Table 4 - Development status of Semantic Population Engine

A.1.3 Semantic Knowledge Base

Module name	RDF Triple Store
GitHub location	N/A
Downloadable artifacts	Knowledge Base Docker image
Development status	N/A
Deployment status	Deployed on a cloud testbed in a Docker container
Integration status	Integrated with the Semantic Reasoner on a cloud testbed.

Dependencies	N/A
Next steps	N/A

Table 5 - Development status of RDF Triple Store

Module name	Domain Ontologies
GitHub location	https://github.com/SODALITE-EU/semantic-models
Development status	Released version - v0.3.0
Deployment status	Deployed on a cloud testbed in a Docker container
Integration status	Ontologies have been imported into the RDF triple store, implementing the conceptual model of SODALITE
Integration issues/dependencies	N/A. There are no dependencies. Ontologies need to be inline with the conceptual model of SODALITE and the Semantic Population Engine (that is responsible for populating the triple store with instances)
Next steps	Updates on the vocabulary, according to the modelling requirements

Table 6 - Development status of Domain Ontologies

A.1.4 Development plan for Semantic Modelling Layer

The development plan is presented in the following table.

M12	M18	M24	M30	M36
SODALITE IDE				
- AADM and RM DSLs - AADM and RM textual editors - AADM and RM save in KB - Basic content assistance	- Extended AADM and RM DSLs - AADM visual representation - Editor content assistance - Syntactic and semantic validation - IDE configuration - AADM	- Model Governance View - Model CRUD in KB - AAI for secure interoperability - Ansible DSL and Editor - Support for policies - Support for modules - AADM outline	- Deployment Governance View - AADM Visual Modeling - Ansible integration - Improved of existing features - Further integration with SODALITE layer	- Improvement of existing features - Further integration with SODALITE layer

	creation wizard - AADM deployment wizard - Optimisation DSL and textual editor - IDE tutorial - IDE CI/CD			
Semantic Reasoner				
- Population of KB with application instances - User context-assistance v1 - Validation of models v1	- Population of KB with infrastructure instances - Optimisation DSL suggestions v1 - User context-assista nce v2 - Making DSL more abstract - Integration with Bug Predictor	- User Management - Workspaces - Tosca Policies - Validation of models v2 - User context-assistan ce v3 - Optimisation DSL suggestions v2 - Integration with Platform Discovery Service	-Runtime optimisation suggestions - User context-assistan ce v4 - Validation of models v3 - Versioning -Suggestions of fixes for tosca and Ansible smells	- Evaluation of the ontology - Advanced reasoning and discovery

Table 7 - Development plan for Semantic Modelling Layer

Appendix A.2 SODALITE Infrastructure as Code Layer

A.2.1 Abstract Model Parser

Module name	Iac-Blueprint-Builder
GitHub location	https://github.com/SODALITE-EU/iac-blueprint-builder
Documentation	README in the repository https://github.com/SODALITE-EU/iac-blueprint-builder/blob/master/README.md
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/iac-blueprint-builder

Development status	Code has been refactored and the issues related to the output have been solved. Minor bugs have also been fixed and the optimisation integration implemented.
Deployment status	Automatically deployed on the testbed using xOpera lightweight orchestrator and the deployment blueprint through Jenkins CI/CD process
Integration status	Integrated with Iac Blueprint Builder, and Iac Resources Model
Dependencies	Python , Flask, flask-swagger-ui, xOpera, Ansible, Docker
Next steps	Authorization/Authentication and more unit tests/

Table 8 - Development status of Abstract Model Parser

A.2.2 IaC Blueprint Builder

Module name	Iac-Blueprint-Builder
GitHub location	https://github.com/SODALITE-EU/iac-blueprint-builder
Documentation	README in the repository https://github.com/SODALITE-EU/iac-blueprint-builder/blob/master/README.md
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/iac-blueprint-builder
Development status	Code has been refactored and the issues related to the output have been solved. Minor bugs have also been fixed and the optimisation integration implemented.
Deployment status	Automatically deployed on the testbed using xOpera lightweight orchestrator and the deployment blueprint through Jenkins CI/CD process
Integration status	Integrated with Sodalite IDE, Abstract Model Parser, Image Builder, IaC Resources Model, and Application Optimiser.
Dependencies	Python , Flask, flask-swagger-ui, xOpera, Ansible, Docker
Next steps	Authorization/Authentication and more unit tests.

Table 9 - Development status of IaC Blueprint Builder

A.2.3 Runtime Image Builder

Module name	Image-builder
GitHub location	https://github.com/SODALITE-EU/image-builder
Documentation	Extensive README in the repository https://github.com/SODALITE-EU/image-builder/blob/master/README.md
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/image-builder-api https://hub.docker.com/r/sodaliteh2020/image-builder-cli
Development status	M12: ● Initial commit ● Flask REST API with Swagger UI for testing ● Dockerized ● deployed to testbed M18: ● Added production security configuration with nginx as reverse proxy ● allowed configuration for host and port through ENV ● TOSCA 1.3 support for deployment to testbed ● Added extensive tests ● Support building image variants with layered base containers ● Add support for the EdgeTPU exporter to the Vehicle IoT UC ● improved documentation M24: ● Added support for SonarCloud QA coverage ● Core tests for pytest integration ● Several SonarCloud Suggestion fixed ● Replaced nginx with traefik for easier container management and integration ● upgraded CI/CD process
Deployment status	Automatically deployed on the testbed using xOpera lightweight orchestrator and the deployment blueprint through Jenkins CI/CD process
Integration status	Integrated with Image Registry, missing integration with IDE
Dependencies	Python , Flask, xOpera, Ansible, Docker, Concrete Image Builder

Next steps	Integration with IDE, add multi-architecture build variants

Table 10 - Development status of Runtime Image Builder

A.2.4 Concrete Image Builder

Module name	Image-builder
GitHub location	https://github.com/SODALITE-EU/image-builder
Documentation	Extensive README in the repository https://github.com/SODALITE-EU/image-builder/blob/master/README.md
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/image-builder-api https://hub.docker.com/r/sodaliteh2020/image-builder-cli
Development status	M12: ● Initial commit ● Flask REST API with Swagger UI for testing ● Dockerized ● deployed to testbed M18: ● Improved workflows ● support for TAR sources ● TOSCA 1.3 support in the workflows ● Added extensive tests ● Support building image variants with layered base containers ● Add support for the EdgeTPU exporter to the Vehicle IoT UC M24: ● No changes in the workflows
Deployment status	Automatically deployed on the testbed using xOpera lightweight orchestrator and the deployment blueprint through Jenkins CI/CD process
Integration status	Integrated with Image Registry, missing integration with IDE
Dependencies	Python , Flask, xOpera, Ansible, Docker, Runtime Image Builder
Next steps	Integration with IDE, add multi-architecture build variants

Table 11 - Development status of Concrete Image Builder

A.2.5 Application Optimiser

Module name	Application Optimiser - MODAK
GitHub location	https://github.com/SODALITE-EU/application-optimisation
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/modakopt/modak
Development status	After an initial preparation phase of the package during the first year of the project (see D3.3), in the second year we have further extended MODAK to support HPC systems cases. In particular, we have prototyped AI training and inference and traditional HPC applications. Currently, MODAK supports TensorFlow, PyTorch, MXnet, mpich, openmpi, and containers for x86 and NVIDIA GPUs. These containers are further labelled with version requirements and support for optimisations like graph compilers or BLAS/LAPACK.
Deployment status	Deployed on the Cloud testbed
Integration status	Partially integrated into the stack Integrated with IaC Blueprint Builder
Dependencies	Standalone service Depends on the IaC Model repository
Next steps	● Complete integration in the SODALITE framework ● Include the Tuner and Scaler features, and extend the performance model to include GPU execution ● Support of Cloud systems ● Support for big data analytics applications

Table 12 - Development status of Application Optimiser - MODAK

A.2.6 IaC Verifier

Module name	IaC Verifier
GitHub location	https://github.com/SODALITE-EU/verification
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/iacverifierapi

Development status	REST API for both Topology Verifier and Provisioning Workflow Verifier
Deployment status	Deployed locally and in the Cloud Testbed
Integration status	Integrated with Topology Verifier and Provisioning Workflow Verifier
Dependencies	Depends on Topology Verifier and Provisioning Workflow Verifier
Next steps	Integration with IDE and IaC Builder

Table 13 - Development status of IaC Verifier

A.2.7 Verification Model Builder

Module name	Verification Model Builder
GitHub location	https://github.com/SODALITE-EU/verification
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/workflowverifie r
Development status	Build models required for checking TOSCA files against the constraints defined by the TOSCA standard. Build PetriNet models from the description files in Petri Net Markup Language (PNML)
Deployment status	Deployed locally and in the Cloud Testbed
Integration status	Integrated with Topology Verifier and Provisioning Workflow
Dependencies	None
Next steps	Update the models used by the Topology Verifier and Provisioning Workflow to support more verification cases

Table 14 - Development status of Verification Model Builder

A.2.8 Topology Verifier

Module name	Topology Verifier

GitHub location	https://github.com/SODALITE-EU/verification
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/toscasynverifier
Development status	The verification of TOSCA models for syntax errors and non-compliance to the standard (1.2) has been implemented.
Deployment status	Deployed locally and in the Cloud Testbed
Integration status	Integrated with Verification Model Builder and IaC Verifier
Dependencies	Depends on Verification Model Builder
Next steps	Checking the compliance of TOSCA models to the newest TOSCA standard (1.3)

Table 15 - Development status of Topology Verifier

A.2.9 Provisioning Workflow Verifier

Module name	Provisioning Workflow Verifier
GitHub location	https://github.com/SODALITE-EU/verification
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/workflowverifie r
Development status	The basic support for translating the control flow model of the Ansible plays to Petri Net Markup Language (PNML) models, and verification of those PNML models
Deployment status	Deployed locally and in the Cloud Testbed
Integration status	Integrated with Verification Model Builder
Dependencies	This component depends on the Verification Model Builder, and is used by IaC Verifier.
Next steps	Improved support for the verification of the control flow of Ansible playbooks

Table 16 - Development status of Deployment Exporter

A.2.10 Bug Predictor and Fixer

Module name	Bug Predictor and Fixer
GitHub location	https://github.com/SODALITE-EU/defect-prediction
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/toscasmells https://hub.docker.com/r/sodaliteh2020/ansiblesmells
Development status	Three types of taxonomies for IaC : best/bad practices taxonomy, smell taxonomy, and bug taxonomy. The support for detecting TOSCA smells with semantic reasoning, Ansible smells with rule-based reasoning, and Ansible linguistic anti-pattern with NLP and deep learning.
Deployment status	Deployed locally and in the Cloud Testbed
Integration status	Integrated TOSCA smell detection capability with IDE and Semantic Reasoner
Dependencies	Depends on Predictive Model Builder and IaC Quality Assessor
Next steps	● Integrate Ansible smell detection with IDE ● Misconfiguration Detection in Ansible playbooks ● Improved support for suggesting fixes for TOSCA smells

Table 17 - Development status of Bug Predictor and Fixer

A.2.11 Predictive Model Builder

Module name	Predictive Model Builder
GitHub location	https://github.com/SODALITE-EU/defect-prediction
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/toscasmells https://hub.docker.com/r/sodaliteh2020/ansiblesmells
Development status	A semantic rule based model was built to detect the security smells in TOSCA. An informal rule based model

	(Ansible-Lint Rules) was built to detect the code and security smells in Ansible. The deep learning and NLP based models were built to detect linguistic anti-patterns in Ansible.
Deployment status	Deployed locally and in the Cloud Testbed
Integration status	Integrated with Bug Predictor and Fixer
Dependencies	Depends on Semantic Reasoner/knowledgebase
Next steps	Adding more deep learning/machine learning models

Table 18 - Development status of Predictive Model Builder

A.2.12 IaC Quality Assessor

Module name	IaC Quality Assessor
GitHub location	https://github.com/SODALITE-EU/iac-quality-framework
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/iacmetrics https://pypi.org/project/ansiblemetrics/
Development status	Implementation for a set of the metrics for Ansible. A REST API
Deployment status	Deployed locally and in the Cloud Testbed
Integration status	Partially integrated Used by Ansible smell detection
Dependencies	None
Next steps	● Add workflow level metrics for Ansible playbooks ● Integrate with IDE to show the quality metrics of IaC developed

Table 19 - Development status of IaC Quality Assessor

A.2.13 IaC Model Repository

Module name	IaC Model Repository

GitHub location	https://github.com/SODALITE-EU/application-optimisation
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/modakopt/modak
Development status	The Model Repository is implemented as a MySQL database. Tables are organized for the systems and applications.
Deployment status	Deployed on the Cloud testbed
Integration status	Partially integrated into the stack Integrated with MODAK
Dependencies	IaC Model Repository interacts with the SODALITE IDE and contains the Performance Model of infrastructure and application (offline analysis).
Next steps	● Develop the schema for the Model Repository and define API for accessing the model repository; ● Develop the full Application and performance Model.

Table 20 - Development status of IaC Model Repository

A.2.14 Image Registry

Module name	Image Registry
GitHub location	https://github.com/SODALITE-EU/iac-platform-stack
Documentation	README in the SODALITE stack deployment repository - The creation of the private Image Registry is implemented through TOSCA Blueprints
Downloadable artifacts	https://hub.docker.com/_/registry/
Development status	M12 ● Initial creation of the private Docker registry ● initial creation repository https://github.com/SODALITE-EU/iac-management/tree/master/blueprint-samples/blueprints/docker-registry ● M18

	● creation and configuration included in the iac-platform-stack M24 ● Improved handling of configuration and cert management
Deployment status	Deployed on the SODALITE testbed and included in the SODALITE stack deployment blueprint
Integration status	Integrated with Image Builder and Orchestrator components
Dependencies	None
Next steps	Improve secure cert handling

Table 21 - Development status of Image Registry

A.2.15 Platform Discovery Service

Module name	Platform Discovery Service
GitHub location	https://github.com/SODALITE-EU/platform-discovery-service
Documentation	README in the GitHub repository
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/platform-discovery-service
Development status	M18 ● initial setup and architecture (internal - not on GitHub) M24 ● xOpera usage and setup ● Creation of the Ansible collections for HPC discovery (Slurm/Torque) ● Ansible playbooks supporting discovery ● Ansible playbooks supporting TOSCA transformation using jinja ● improved tests ● SonarCloud integration and coverage
Deployment status	Deployed on the SODALITE testbed and included in the SODALITE stack deployment blueprint
Integration status	Deployed on the Testbed and integrated in the blueprint. Not fully integrated with IDE, not fully integrated with Semantic-Reasoner.

Dependencies	Depends on the inputs from the IDE component provided by the Resource Expert
Next steps	Integration with IDE, integration with Semantic Reasoner

Table 22 - Development status of Platform Discovery Service

A.2.16 Development plan for Infrastructure as Code Management layer

The development plan is presented in the following table.

M12	M18	M24	M30	M36
Abstract Model Parser and IaC Blueprint Builder				
Initial commit, Flask REST API with Swagger UI for testing, Dockerized deployed to testbed	Making the parser work to produce TOSCA blueprints from AADM JSON. Added some unit tests. Integration with IDE. Integration with other components required. Added support for SonarCloud QA coverage. Core tests for pytest integration.	Refactored the code. Solved several bugs. Worked on the issues for the wrong output. Made relevant changes according to the change in AADM JSON. Several SonarCloud suggestions fixed. Added integration with MODAK. Some new tests were added.	Add more unit tests. Add authentication/ authorization for the security of the component as required.	Improve code efficiency for better maintainability.
Runtime Image Builder and Concrete Image Builder				
Initial commit, Flask REST API with Swagger UI for testing, Dockerized deployed to testbed	Added production security configuration with nginx as reverse proxy, allowed configuration for host and port through ENV TOSCA 1.3, support for	Added support for SonarCloud QA coverage. Core tests for pytest integration, Several SonarCloud suggestions fixed, Replaced nginx with traefik for	Integration with IDE, add multi-architecture build variants	Improved workflows for the image building process

	deployment to testbed, Added extensive tests, Support building image variants with layered base containers, Addes support for the EdgeTPU exporter to the Vehicle IoT UC, improved documentation,	easier container management and integration, upgraded CI/CD process.		
Image Registry				
Initial creation of the private Docker registry	creation and configuration included in the iac-platform-sta ck	Improved handling of configuration and cert management	N/A	N/A
Application Optimiser				
Performance model	AI Training and HPC parallel applications support	HPC-systems support	Cloud-system support	Big-data analytics application support
IaC Taxonomies				
IaC Smell Taxonomy (Initial Version)	IaC Best and Bad Practices Taxonomy (Initial Version)	IaC Best and Bad Practices Taxonomy (Final Version), IaC Smell Taxonomy (Final Version) IaC Bug Taxonomy (Initial Version)	IaC Bug Taxonomy (Final Version)	Validated Final Taxonomies
IaC Bug Prediction and Correction, Verification, and Quality Assurance				

Smell Detection in TOSCA and Ansible (Basic Support)	Integration of TOSCA Smell Detection with IDE, Structural and Control Flow Verification of IaC (Basic Support), Linguistic Anti-Pattern Detection of Ansible (Basic Support)	Smell Detection TOSCA and Ansible (Improved Support), Linguistic Anti-Pattern Detection in Ansible (Improved Support), Structural Ansible Metrics	Suggesting for Fixes for TOSCA Smells (Basic Support), Misconfiguration Detection for Ansible, Control Flow Ansible Metrics, Control Flow Verification of Ansible (Improved Support)	Suggesting for Fixes for TOSCA Smells (Improved Support), Improved Integration of Bug/Smell Detection and Verification with IDE
Platform Discovery Service				
	initial setup and architecture (internal - not on GitHub)	Initial Discovery for Openstack, Torque, Slurm, AWS Ansible collections supporting HPC, additional Ansible collections supporting Openstack security groups and keypairs	Initial integration with Semantic Reasoner, Improved integration with IDE, Initial kubernetes discovery	Support for TOSCA changes, Improved versions of TOSCA for Openstack, AWS and kubernetes discovery

Table 23 - Development plan for IaC Management Layer

Appendix A.3 SODALITE Runtime Layer

A.3.1 Orchestrator + Drivers

Orchestrator

Module name	Orchestrator - xOpera
GitHub location	https://github.com/xlab-si/xopera-opera
Documentation	https://xlab-si.github.io/xopera-opera/
Downloadable artifacts	https://pypi.org/project/opera/

Development status	xOpera is a lightweight orchestrator compliant with OASIS TOSCA standard. The current compliance is with the TOSCA Simple Profile in YAML v1.3. It is mostly developed by XLAB although any contribution is welcome. Since the initial version, xOpera has been further developed by RADON and SODALITE projects making contributions.
Deployment status	Currently the xOpera version used is 0.6.4 which is deployed end encapsulated as a dockerized REST API deployed on the testbed.
Integration status	Used as a dockerized REST API in the SODALITE deployment pipeline, integrated with IDE and iac-blueprint-builder. Deploys exporters needed for monitoring through TOSCA/Ansible blueprints.
Dependencies	Python, Ansible, Openstack
Next steps	Improve redeployment, add TOSCA policy enforcement implementation, improve secret handling

Table 24 - Development status of Orchestrator - xOpera

ALDE

Module name	ALDE
GitHub location	https://github.com/SODALITE-EU/alde
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/alde
Development status	ALDE has been developed to its requirements scope.
Deployment status	ALDE is deployed in the Orchestrator environment as part of the SODALITE Stack
Integration status	ALDE is integrated with a HPC environment and it is capable of executing Singularity containers through a REST API interface.
Dependencies	HPC environment
Next steps	No further steps required

Table 25 - Development status of ALDE

A.3.2 Monitoring

Monitoring system

Module name	Monitoring - Prometheus
GitHub location	https://github.com/prometheus/prometheus[33]
Documentation	https://prometheus.io/
Downloadable artifacts	https://hub.docker.com/r/prom/prometheus
Development status	N/A (third party baseline component integrated in SODALITE Runtime Layer)
Deployment status	Prometheus is deployed in a Docker container as part of the SODALITE Stack: https://github.com/SODALITE-EU/iac-platform-stack/blob/baec5ccd39d4d5f3a31815eed94dcfdb9ba8ddfd/docker-local/service.yaml#L621
Integration status	Monitoring is integrated with Consul and Alert Manager. Using Consul, Monitoring request metrics to registered exporters: Node, IPMI, HPC, Skydive
Dependencies	Monitoring requires Consul
Next steps	No further steps required

Table 26 - Development status of Monitoring - Prometheus

Module name	Monitoring - Consul
GitHub location	https://github.com/hashicorp/consul[34]
Documentation	https://www.consul.io/docs
Downloadable artifacts	https://hub.docker.com/_/consul
Development status	N/A (third party baseline component integrated in SODALITE Runtime Layer)
Deployment status	Consul is deployed in a Docker container as part of the SODALITE Stack: https://github.com/SODALITE-EU/iac-platform-stack/blob/master/docker-local/service.yaml#L585

Integration status	Consul is connected to Prometheus. Several exporters register themselves in Consul: Node, IPMI, HPC, Skydive
Dependencies	None
Next steps	No further steps required

Table 27 - Development status of Monitoring - Consul

Alert Manager

Module name	Alert Manager
GitHub location	https://github.com/SODALITE-EU/monitoring-system/tree/master/ruleserver
Documentation	https://github.com/SODALITE-EU/monitoring-system/blob/master/ruleserver/README.md
Downloadable artifacts	https://hub.docker.com/r/prom/alertmanager
Development status	Initial version: registration in Prometheus, reception of monitoring alerts
Deployment status	Alert Manager is deployed in a Docker container as part of the SODALITE Stack: https://github.com/SODALITE-EU/iac-platform-stack/blob/7b65739d4ff5c334654ec9c1848ee6ff131b62ca/docker-local/service.yaml#L597
Integration status	Alert Manager is registered within Monitoring - Prometheus
Dependencies	Python, Flask, Gunicorn
Next steps	- API for managing subscribers - implementation of scaling or reconfiguration policies

Table 28 - Development status of Alert Manager

Monitoring Dashboard - Grafana

Module name	Monitoring Dashboard - Grafana
GitHub location	https://github.com/grafana/grafana[35]

Documentation	https://grafana.com/docs/
Downloadable artifacts	https://grafana.com/grafana/download
Development status	Single instance installed in Atos testbed Some dashboards configured for monitoring data from monitoring scraped by Node and HPC Exporters
Deployment status	Manual deployment in Atos testbed
Integration status	Manual integration with Monitoring
Dependencies	Prometheus
Next steps	Orchestration based, blueprint mediated automatic deployment within the SODALITE stack Automatic configuration of dashboards for target execution environments and applications

Table 29 - Development status of Monitoring Dashboard

IPMI Exporter

Module name	IPMI Exporter
GitHub location	https://github.com/SODALITE-EU/ipmi-exporter
Documentation	https://github.com/SODALITE-EU/ipmi-exporter/blob/master/README.md
Downloadable artifacts	N/A (component to be deployed by the SODALITE Orchestrator)
Development status	Exporter implemented Registration with Consul
Deployment status	CI/CD building. Manual deployment
Integration status	IPMI Exporter registers itself in consul upon initialization
Dependencies	Go, Consul
Next steps	Automation of the registering and deregistering processes in Consul through Ansible playbooks Automated deployment with Orchestrator

Table 30 - Development status of IPMI Exporter

HPC Exporter

Module name	HPC Exporter
GitHub location	https://github.com/SODALITE-EU/hpc-exporter
Documentation	https://github.com/SODALITE-EU/hpc-exporter/blob/master/README.md
Downloadable artifacts	Docker image for building the container is available at: https://github.com/SODALITE-EU/hpc-exporter/tree/master/docker
Development status	Prometheus exporter for scraping metrics about job execution in HPC clusters. Schedulers supported: - PBS Pro - Slurm
Deployment status	Docker based deployment. Manual deployment
Integration status	HPC Exporter registers itself in consul upon initialization
Dependencies	Go, Consul
Next steps	Queue metrics Automation of the registering and deregistering processes in Consul through Ansible playbooks Automated deployment with Orchestrator

Table 31 - Development status of HPC Exporter

Skydive Exporter

Module name	prometheus-skydive-connector
GitHub location	https://github.com/skydive-project/skydive-flow-exporter/tree/master/prom_sky_con
Documentation	https://github.com/skydive-project/skydive-flow-exporter/blob/master/prom_sky_con/README.md http://skydive.network/blog/prometheus-connector.html

Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/prometheus-skydive-connector
Development status	Basic connector implemented. Pushed to upstream. Registration with Consul.
Deployment status	Deployed in a Docker container or manually. Incorporated into SODALITE platform blueprint.
Integration status	Registers in consul via platform stack
Dependencies	Skydive, Prometheus
Next steps	Extend types of metrics collected.

Table 32 - Development status of Skydive Exporter

A.3.3 Deployment Refactorer

Module name	Deployment Refactorer
GitHub location	https://github.com/SODALITE-EU/refactoring-ml
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/rule_based_refactorer https://hub.docker.com/r/sodaliteh2020/fo_perf_predictor_api
Development status	We developed the component and demonstrated its practicality and feasibility by applying it to 1) an extension of the RuBiS benchmark application[36][37] deployed on Google's Compute Engine (92 deployment alternatives), 2) Teastore microservice benchmark application on Google Kubernetes Engine (78 deployment alternatives) and SODALITE snow use case. We developed the rule-based adaptation support, and validated it with three key scenarios in the Vehicle IoT use case: location-aware redeployment, alert-driven redeployment: Cloud alerts, alert-driven redeployment: Edge alerts. We have been experimenting with a recent microserice memory anomaly dataset[38] using three predictive algorithms Random Forest, Decision Tree and DL with

	AdaBoost. We have started to collect a service network anomaly dataset with TeaStore microservice benchmark[39], Google kubernetes Engine, and SODALITE SkyDive monitoring support.
Deployment status	Deployed locally and in the Cloud Testbed
Integration status	Integrated with Orchestrator, Monitoring Layers, and Refactoring Option Discoverer
Dependencies	This module depends on Refactoring Option Discoverer, Node Manager, Deployment Preparation API, Monitoring Layer
Next steps	● Integrate with Node Manager ● Complete performance anomaly detection support ● Support all deployment refactoring scenarios in SODALITE use cases.

Table 33 - Development status of Deployment Refactorer

A.3.4 Node Manager

Module name	reactoring-ct
GitHub location	https://github.com/SODALITE-EU/refactoring-ct
Documentation	https://github.com/SODALITE-EU/refactoring-ct/blob/master/README.md
Downloadable artifacts	N/A
Development status	Implementation completed
Deployment status	Deployed on a private cloud. The component is ready to be deployed in the testbed. TOSCA blueprint to be completed.
Integration status	Missing integration with the SODALITE Monitoring infrastructure and with Deployment Refactorer
Dependencies	Python, Kubernetes
Next steps	Integration with SODALITE Monitoring and Deployment Refactorer

Table 34 - Development status of Node Manager

A.3.5 Refactoring Option Discoverer

Module name	Refactoring Option Discoverer
GitHub location	https://github.com/SODALITE-EU/refactoring-option-discoverer
Documentation	README in the repository
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/refactoring_option_discoverer
Development status	Can discover compute and software nodes by matchmaking on node properties, node capabilities and requirements, and node policies (TOSCA)
Deployment status	Deployed locally and in the Cloud Testbed
Integration status	Integrated with Deployment Refactorer and Knowledge Base/Semantic Reasoner.
Dependencies	Depends on the Semantic Knowledge Base/Semantic Reasoner
Next steps	Improve the existing matchmaking capabilities to support all dynamic resource discovery and composition requirements by SODALITE case studies

Table 35 - Development status of Refactoring Option Discoverer

A.3.6 xOpera REST API

Module name	xopera-rest-api
GitHub location	https://github.com/SODALITE-EU/xopera-rest-api
Documentation	Extensive README in the repository https://github.com/SODALITE-EU/xopera-rest-api/blob/master/README.md
Downloadable artifacts	https://hub.docker.com/r/sodaliteh2020/xopera-rest-api
Development status	Stable, adding functional requirements from SODALITE created in GitHub issues

Deployment status	Automated Deploy on the testbed staging environment through xOpera and TOSCA blueprints; implemented in the Jenkins CI/CD workflow
Integration status	Used as a dockerized REST API in the SODALITE deployment pipeline (through IDE and iac-blueprint-builder); deploys exporters needed for monitoring through TOSCA/Ansible blueprints - missing Adding rules to AlertManager through TOSCA blueprints, integration with node-manager for deploying applications
Dependencies	Python, Flask, connexion, Ansible, OpestackAPI, boto, boto3,
Next steps	support for OpenFaaS, improved interface covering newly introduced commands in xOpera for feature support, improved security

Table 36 - Development status of xOpera REST API

A.3.7. Kubernetes Edge Components

Kubernetes Edge components are split into multiple categories (labeller, controllers/monitors, device plugins, and exporters). While these all form a part of the Edge deployment, most of these components are general Kubernetes controllers and have no specific dependence on the Edge. A number of upstream components are also used, which have been extended for the Edge case.

Controllers / Monitors

Module name	k8s-node-label-monitor
GitHub location	https://github.com/adaptant-labs/k8s-node-label-monitor
Documentation	https://github.com/adaptant-labs/k8s-node-label-monitor/blob/master/README.md
Downloadable artifacts	https://github.com/adaptant-labs/k8s-node-label-monitor/releases
Development status	Initial implementation done. Should be extended for annotation monitoring in order to provide notice of device plugin resource allocation/availability changes.
Deployment status	Deployed on Edge Testbed. Can be run as a standalone Docker image, or installed into a Kubernetes cluster directly via manifest or Helm chart.
Integration status	Part of Edge stack, but not Edge or HW-specific.

Dependencies	Go, Kubernetes
Next steps	Extend for annotation monitoring.

Table 37 - Development status of K8s Node Label Monitor

Labellers

Module name	k8s-dt-node-labeller
GitHub location	https://github.com/adaptant-labs/k8s-dt-node-labeller
Documentation	https://github.com/adaptant-labs/k8s-dt-node-labeller/blob/master/README.md
Downloadable artifacts	https://github.com/adaptant-labs/k8s-dt-node-labeller/releases
Development status	Initial implementation done.
Deployment status	Deployed on Edge Testbed. Can be run as a standalone Docker image, or installed into a Kubernetes cluster directly via manifest or Helm chart.
Integration status	Part of Edge stack, Edge-specific.
Dependencies	Go, Kubernetes
Next steps	None

Table 38 - Development status of K8s DeviceTree Node Labeller

Module name	k8s-auto-labeller
GitHub location	https://github.com/adaptant-labs/k8s-auto-labeller
Documentation	https://github.com/adaptant-labs/k8s-auto-labeller/blob/master/README.md
Downloadable artifacts	https://github.com/adaptant-labs/k8s-auto-labeller/releases
Development status	Initial implementation done. Label aliases and sources will need to be updated for matching the final deployment configuration.

Deployment status	Deployed on Edge Testbed. Can be run as a standalone Docker image, or installed into a Kubernetes cluster directly via manifest or Helm chart.
Integration status	Part of Edge stack, but not Edge or HW-specific.
Dependencies	Go, Kubernetes
Next steps	Extend labels to match final HW and deployed device plugin configuration.

Table 39 - Development status of K8s Node Auto Labeller

Device Plugins

A number of accelerator-specific device plugins are installed into the Kubernetes cluster in order to provide device management and resource allocation controls for Kubernetes pod scheduling. Both the EdgeTPU and NVIDIA GPUs have pre-existing device plugins which have been extended for the Edge case, and can be used directly. No device plugin for the Intel NCS2 accelerator existed, so one was developed:

Module name	ncs2-device-plugin
GitHub location	https://github.com/adaptant-labs/ncs2-device-plugin
Documentation	https://github.com/adaptant-labs/ncs2-device-plugin/blob/master/README.md
Downloadable artifacts	https://github.com/adaptant-labs/ncs2-device-plugin/releases
Development status	Node-label based device information provided, implementation of device plugin API to enable automated resource allocation and limits in progress.
Deployment status	Deployed on Edge Testbed. Can be run as a standalone Docker image, or installed into a Kubernetes cluster directly via manifest or Helm chart.
Integration status	Part of Edge stack.
Dependencies	Python, Kubernetes, Intel NCS2 accelerator
Next steps	Extension for Kubernetes device plugin API

Table 40 - Development status of Intel NCS2 Device Plugin

Exporters

Note that each Edge Gateway is provisioned with its own set of exporters. By default, the Kubernetes instance of Prometheus and AlertManager is cluster-wide, and a Node Exporter instance is automatically scheduled on any node joining the cluster. In addition to the Node Exporter, a number of accelerator-specific exporters are scheduled on any Edge nodes that satisfy the affinity constraints (e.g. dependency labels and annotations as exposed by the device plugins).

Module name	prometheus_ncs2_exporter
GitHub location	https://github.com/adaptant-labs/prometheus_ncs2_exporter
Documentation	https://github.com/adaptant-labs/prometheus_ncs2_exporter/blob/master/README.md
Downloadable artifacts	https://github.com/adaptant-labs/prometheus_ncs2_exporter/releases
Development status	Basic exporter implemented. The NCS2 device requires a valid ML model to be loaded before the temperature can be read, so any service wishing to monitor the temperature will need to provide an instantiation of the exporter itself - an API is provided for this, and the steps are documented in the README.
Deployment status	Deployed on Edge Testbed. Can be run as a standalone Docker image, or installed into a Kubernetes cluster directly via manifest or Helm chart.
Integration status	No special configuration is required in Kubernetes, exporter is automatically scraped by Prometheus. In order for the thermal metric to be exposed, it must be instantiated within an inference application.
Dependencies	Python, Kubernetes, Prometheus, Intel NCS2 accelerator
Next steps	Integration with inference application

Table 41 - Development status of Intel NCS2 Exporter

Module name	edgetpu-exporter
GitHub location	https://github.com/adaptant-labs/edgetpu-exporter
Documentation	https://github.com/adaptant-labs/edgetpu-exporter/blob/master/README.md

Downloadable artifacts	N/A
Development status	Basic exporter implemented.
Deployment status	Deployed on Edge Testbed. Can be run as a standalone Docker image, or installed into a Kubernetes cluster directly via manifest or Helm chart.
Integration status	No special configuration is required in Kubernetes, exporter is automatically scraped by Prometheus.
Dependencies	Go, Kubernetes, Prometheus, EdgeTPU accelerator
Next steps	None

Table 42 - Development status of Google Coral EdgeTPU Exporter

A.3.7 Development plan for Runtime Layer

The development plan is presented in the following table.

M12	M18	M24	M30	M36
Orchestrator				
Deployment and provisioning of Openstack and dockerized components, initial iac modules covering Torque deployment	Merge imported topology templates, Parallel task execution, Deployment Resume and force start feature, Added tests and integration tests, Better support for deployment of TOSCA CSAR	Implementation of Diff and Update commands fully supporting partial redeployment Covered TOSCA policy definition, intrinsic function support, Python library API, Improved documentation,	Enforcement of TOSCA policies, security, improved parser validation output, Improved support for reconfiguration scenarios - update method in interface that would allow not to remove instances	Improved handling of secrets through local Ansible Vault , support TOSCA 2.0 simple yaml standard
Monitoring				
Monitoring instance, Node Exporter	Edge Exporter, IPMI Exporter, Monitoring Dashboard	Consul, Monitoring integration with Consul, HPC Exporter,	Alert Manager subscription API, Scaling and configuration alert	Automatic dashboard configuration, Monitoring of HPC queues

		Skydive Exporter, Alert Manager, Alert rules	management, Automatic deployment by Orchestrator (monitoring, dashboard, exporters), automatic registration/deregistration of exporters in Consul	
Deployment Refactorer				
Basic rule-based refactoring decision making, Integration with Orchestrator, A single machine learning algorithm for predicting performance	Integration with Monitoring Layer, More improved machine learning algorithms	Improved rule-based refactoring with high-level support, Complete Performance Prediction Machine Learning Pipeline, Switching between Alternative Deployments (Basic support), Integration with Refactoring Option Discoverer	Performance Anomaly Detection, Switching between Alternative Deployments (Improved support), Support most Refactoring Scenarios in SODALITE Use Cases	Integration with Node Manager
Node Manager				
Basic GPU/CPU loadbalancing, Control theoretical planners, Monitoring	Supervisor	Smart GPU/CPU loadbalancer, Integration in Kubernetes, Support for TOSCA inputs	Deployment in the SODALITE infrastructure, Integration with Monitoring	Integration with Deployment Refactorer
Refactoring Option Discoverer				
Matchmaking based on	Matchmaking based on	Matchmaking based Node	Matchmaking based Node	Support for Dynamic

constraints on Node properties (Basic Support)	constraints on Node properties (Improved Support)	Policies (Basic Support), Matchmaking based Node Requirements and Capabilities (Basic Support)	Policies (Improved Support), Matchmaking based Node Requirements and Capabilities (Improved Support)	Resource Discovery Scenarios in SODALITE Use Cases
		xOpera REST API		
Initial dockerized REST API release covering Torque and Openstack deployments, Initial handling of deployment session and states, Initial partial redeployment	Support for xOpera 0.5.7 features, Included plugin for Git handling blueprint deployments shared among users, Support TOSCA 1.3 yaml version, corrected access to Openstack now possible through environment variables, Additional JWT support	Improved Git handling plugin, Using xOpera as API calls, Moved to connexion for easier REST API design handling, replaced reverse proxy form nginx to traefik (for a simplified Docker container detection and management), implemented support for Kubernetes support for AWS collections, support for IAM token authentication through introspection, support for handling secrets, support for redeployment through *diff* and *update* xOpera commands	Support for OpenFaaS, Improved secret handling, REST API support for newly added commands in xOpera, Improved integration with refactoring, Improved support for Kubernetes, support for GoogleCloud	Improved support for secure storage handling, REST API support for newly added commands in xOpera
		Kubernetes Controller for Edge		
Initial Edge Exporter implementation	Resource & Capabilities discovery,	Expanded run-time monitoring of	Dynamic alerting rules matching Edge node	Per-node managed / scoped

	node labelling.	node annotations, integration with accelerator device plugins. Self-contained instantiation of Edge nodes via Operator SDK.	profile. Dynamic addition and removal of heterogeneous accelerators. Pod/Deployment migration between accelerator types.	deployments, scaling (allowing SODALITE to refactor a node-specific instance of a deployment, without impacting deployments on other nodes in the cluster).

Table 43 - Development plan for Runtime Layer

Appendix A.4 SODALITE Security Components

A.4.1 IAM Introspection

Module name	IAM Introspection - Keycloak
GitHub location	https://github.com/keycloak/keycloak
Documentation	https://www.keycloak.org/documentation.html
Downloadable artifacts	https://hub.docker.com/r/jboss/keycloak
Development status	N/A (third party baseline component integrated in SODALITE platform)
Deployment status	Deployed in the Cloud testbed
Integration status	Integrated with IDE, Semantic Reasoner, Platform Discovery Service, xopera-rest-api, ...
Dependencies	
Next steps	Integrate with IaC Management and Runtime Layer

Table 44 - Development status of IAM Introspection

A.4.2 Secrets Management

Module name	Secrets Management - HashiCorp Vault
GitHub location	https://github.com/hashicorp/vault

Documentation	https://www.vaultproject.io/docs
Downloadable artifacts	https://hub.docker.com/_/vault
Development status	N/A (third party baseline component integrated in SODALITE platform)
Deployment status	Deployed in the Cloud testbed
Integration status	Integrated with IDE, Semantic Reasoner, Platform Discovery Service, xopera-rest-api, ...
Dependencies	
Next steps	Integrate with IaC Management and Runtime Layer

Table 45 - Development status of Secrets Management