Sodalite

SOftware Defined AppLication Infrastructures managemenT and Engineering

# IaC Management - Intermediate version

D4.2

**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** | D4.2 - IaC Management - intermediate version | | |
| **Authors** | Elisabetta Di Nitto (POLIMI), Emilio Imperiali (POLIMI), Saloni Kyal (POLIMI), Dragan Radolović (XLAB), Alexander Maslennikov (XLAB), Kalman Meth (IBM), Yosu Gorroñogoitia (ATOS), Kamil Tokmakov (USTUTT), Indika Kumara (JADS), Alfio Lazzaro (HPE), Paul Mundt (ADPT) | | |
| **Reviewers** | Yosu Gorroñogoitia (ATOS), Zoe Vasileiou (CERTH) | | |
| **Dissemination level** | Public | | |
| **History of changes** | Name | Change | Date |
| | Elisabetta Di Nitto | Structure definition | 01/10/2020 |
| | All | Structure refinement | 15/10/2020 |
| | All | Preliminary contributions | 30/11/2020 |
| | All | Refinements | 20/12/2020 |
| | All | Restructuring of material | 04/01/2021 |
| | All | Finalization of contributions | 15/01/2021 |
| | Elisabetta Di Nitto | Finalization of draft for internal review | 18/01/2021 |
| | Elisabetta Di Nitto, Nejc Bat | Finalization of document for final submission | 29/01/2021 |

## Acknowledgement

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


*© Copyright Beneficiaries of the SODALITE Project*

# Table of Contents





*© Copyright Beneficiaries of the SODALITE Project*

*© Copyright Beneficiaries of the SODALITE Project*

*© Copyright Beneficiaries of the SODALITE Project*

## List of figures




*© Copyright Beneficiaries of the SODALITE Project*

## Executive Summary

The purpose of this deliverable is to present the status of the IaC Management Layer at the end of the second year of the SODALITE project.

The SODALITE Infrastructure as Code (IaC) Management Layer acts as an intermediary between the modeling environment, which aims at supporting the development of Abstract Application Deployment Models (AADMs), and the runtime environment, where the applications are actually executed. To accomplish its intermediation function, this layer offers several support services concerning the following activities:

- Discovery of resources and automatic generation of Resource Models.
- Generation of the IaC code from the AADM.
- Creation of the container images to support the performance-optimized execution of application components on specific target environments.
- Identification of smells/bugs in the IaC scripts and their prediction and fixing.
- Configuration of the containerized application to achieve an optimized execution in the HPC environment.

In the current deliverable we provide an overview of the current evolution of the IaC Management Layer with respect to the points described above. In particular, this deliverable focuses on the progress of the work with respect to what was reported in Deliverable D4.1 [D4.1] at the end of the first project year. The deliverable has been developed in parallel and coherently to WP2, WP5 and WP6 deliverables [D2.2, D5.2, D6.3, D6.6] and to the work developed in WP3 as part of the second year.

The main innovations accomplished during this year are the following:

- Automated platform discovery: this new mechanism allows the identification of specific resources and the creation of the corresponding Resource Models and TOSCA node types. The approach works by querying the status of the available infrastructures at a certain point in time. This is particularly important as it relieves the Resource Expert (RE) from the need to manually model and define resource types, thus saving significant modeling time.
- Support for the creation of Ansible scripts integrated with the Resource Models: this is an add-on that offers users content assistance mechanisms that guide developers in the creation of scripts that are coherent with the definition of the resource models in which context they are used.
- The MODAK package, a software-defined optimisation framework for containerised HPC and AI applications: this is responsible for enabling the static optimisation of applications before deployment. MODAK aims to optimise the performance of applications deployed to HPC infrastructures in a software-defined way. Automation in application optimisation is enabled by using performance modelling and container technology.
- 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; improved support for detecting smells via a semantic approach.

During this year, the consortium has also consolidated the pre-existing components of the IaC Management Layer, namely, the Image Builder, the IaC Blueprint Builder, the Prediction Services, the IaC Quality Assessors, the Topology Verifier and the Provisioning Workflow Verifier.


*© Copyright Beneficiaries of the SODALITE Project*

A special attention has been posed not only to the development of new features and integration mechanisms for these components, but also to the improvement of their quality and robustness.
All components of the IaC Management Layer, together with the other SODALITE components, have been incorporated into an overall CI/CD process with quality gates both in terms of code analysis (through SonarCloud) and in terms of testing.
In the third project year all IaC Management Layer components will be enriched with additional features.


*© Copyright Beneficiaries of the SODALITE Project*

## Glossary

| Acronym | Explanation |
|---|---|
| **AADM** | Abstract Application Deployment Model used in SODALITE to abstract the application deployment. |
| **AI** | Artificial Intelligence |
| **AOE** | Application Ops Expert<br>The equivalent process from the ISO/IEC/IEEE standard 12207 Systems and software engineering — Software life cycle processes is Operation processes and maintenance processes |
| **API** | Application Program Interface |
| **CPU** | Central Processing Unit |
| **CRESTA** | Collaborative Research into Exascale Systemware, Tools & Applications |
| **DSL** | Domain-Specific Language |
| **EMF** | Eclipse Modeling Framework |
| **ETL** | Extract, Transform, Load |
| **GPU** | Graphical Processing Unit |
| **FPGA** | Field-Programmable Gate Array |
| **HPC** | High Performance Computing |
| **IaC** | Infrastructure as Code |
| **IaaS** | Infrastructure as a Service |
| **IAM** | Identity and Access Management |
| **IDE** | Integrated Development Environment |
| **JSON** | JavaScript Object Notation |
| **k8s** | Kubernetes |
| **MAMBA** | Managed Abstract Memory Arrays |
| **M2T** | Model-to-Text |
| **OASIS** | Organization for the Advancement of Structured Information Standards |
| **QE** | Quality Expert<br>The equivalent process from ISO/IEC/IEEE standard 12207 Systems and software engineering — Software life cycle processes: Infrastructure management and Configuration management processes |
| **QoS** | Quality of Service |
| **RDF** | Resource Description Framework |
| **RE** | Resource Expert<br>The equivalent process from ISO/IEC/IEEE standard 12207 Systems and |


© Copyright Beneficiaries of the SODALITE Project

| | |
|---|---|
| | software engineering — Software life cycle processes is Quality Management and Quality assurance processes |
| **REST** | Representational State Transfer |
| **SHACL** | Shapes Constraint Language |
| **SSH** | Secure Shell |
| **TOSCA** | Topology and Orchestration Specification for Cloud Applications |
| **TLS** | Transport Layer Security |
| **UDJ** | Universal Data Junction |
| **UML** | Unified Modeling Language |


*© Copyright Beneficiaries of the SODALITE Project*

# 1 Introduction

The SODALITE Infrastructure as Code (IaC) Management Layer acts as an intermediary between the modeling environment, which aims at supporting the development of Abstract Application Deployment Models (AADMs), and the runtime environment, where the applications are actually executed. To accomplish its intermediation function, this layer offers several support services concerning the following activities:

- *Discovery of resources and automatic generation of Resource Models*: one of the main problems Application Ops Experts (AOE) face when they want to deploy their applications concerns understanding which kinds of resources they could use and analyzing their characteristics. To accomplish this task, they typically rely on informal descriptions of resources that are provided by the corresponding providers. Such descriptions can be expressed in different formats and can be difficult to compare them. The TOSCA standard [Rutkowski20] is addressing this problem by offering the notion of Node Type and by suggesting resource providers or intermediaries to use it to define models of resources. In SODALITE, the IaC Management Layer moves one step forward and offers a service, the Platform Discovery Service, to automate the definition of Resource Models and corresponding TOSCA Node Types given the resources discovered in a certain infrastructure.
- *Generation of the IaC code from the AADM*: to enable interoperability with other initiatives and projects, SODALITE runtime is centered around the standard language for deployment orchestration, that is, TOSCA. The IaC Management Layer then transforms abstract models defined by the Application Ops Experts into TOSCA executable blueprints. Moreover, it also supports the operationalization of these blueprints by offering support to the creation of Ansible scripts that implement all operations required in a TOSCA blueprint.
- *Creation of the container images to support the performance-optimized execution of application components on specific target environments*: In order to enable the execution of application components on different resources, we encapsulate them in proper containers that create the needed abstraction level between the application code and the actual executing environment. In this context, the IaC Management Layer offers the Image Builder service that addresses the problem of automating the creation of component images so that they can be properly executed within the context of Docker[1] and Singularity[2] containers, as at the moment, these two technologies are the most used application packaging approaches for Cloud and HPC environments, respectively.
- *Identification of smells/bugs in the IaC scripts and prediction and fixing of smells/bugs*: As any other piece of software, IaC can contain smells and bugs. This occurs even when the code is automatically generated, as it happens in SODALITE. Typical problems may concern, for instance, an incorrect definition of the workflow steps in the Ansible components. To address this problem, the IaC Management Layer offers a suite of components constituted by the Predictor Services, the IaC Quality Assessors, the Topology Verifier and the Provisioning Workflow Verifier, as well as a taxonomy of IaC and the corresponding catalogues of smells and bugs.
- *Configuration of the containerized application to achieve an optimized execution in the HPC environment*: AI training frameworks require target-specific libraries and drivers to be configured. In the context of HPC infrastructures, with diverse hardware and software dependencies and libraries, building or selecting an optimised container for deploying

[1] https://www.docker.com/ Package Software into Standardized Units for Development, Shipment and Deployment

[2] https://sylabs.io/singularity/ Users of singularity can build applications on their desktops and run hundreds or thousands of instances—without change—on any public cloud or out to the computational edge.


© Copyright Beneficiaries of the SODALITE Project

AI-based components is crucial. To address this issue, the IaC Management Layer offers an application optimizer called MODAK that maps the optimal application parameters to the infrastructure target by building or selecting an optimised container and then encoding optimisations in a job script.

### 1.1 Deliverable goal

The goal of this deliverable is to provide an overview of the current evolution of the IaC Management Layer with respect to the points described above. In particular, this deliverable focuses on the progress of the work with respect to what was reported in Deliverable D4.1 [D4.1] at the end of the first project year.

Therefore, the deliverable provides updates concerning the services developed in the first project year, that are the Image Builder, the IaC Blueprint Builder, the Prediction Services, the IaC Quality Assessors, the Topology Verifier and the Provisioning Workflow Verifier. Moreover, it offers a detailed overview of the newly added services focusing on the innovations they offer, their main features and architecture.

This deliverable has been developed in parallel and coherently to WP2, WP5 and WP6 deliverables [D2.2, D5.2, D6.3, D6.6] and to the work developed in WP3 as part of the second year.

### 1.2 Structure of the document

Consistently with its goals, the rest of this deliverable is structured in the following main sections:

- Section 2 presents the evolution of the IaC Management Layer architecture. This evolution is described in further details in the requirements and architecture deliverable [D2.2].
- Section 3 briefly presents the development and integration tools exploited in WP4, in agreement with the general rules defined for the SODALITE project, as described in [D2.4].
- Section 4 presents the new features developed in the second project year. As mentioned, we highlight their innovation with respect to the state of the art, the offered features, the architecture of the corresponding components and provide an overview of the current status of development and of the plan for the last project year.
- Section 5 provides a brief update of the status of the components already available in the first year in terms of their evolution in the second project year.
- Section 6 presents the IaC Management Layer development plan for the third year. This plan is an excerpt of the general plan presented in Appendix A of the Deliverable D6.6 [D6.6] and is presented also here for the sake of completeness.
- Section 7 presents the conclusions.
- Finally, the Appendix provides some details concerning the Ansible metamodel (see Section 4.2).

## 2 Changes to the IaC Management Layer Architecture

During the second year of the project, several changes have been made to the IaC management layer architecture. During the intensive phase of component development, integration issues have been identified and therefore addressed reflecting the implementation of the components and their integration into the SODALITE framework. From the perspective of the IaC management layer, the Platform Discovery Service has been added to the framework to enable partial automation for the Resource Expert (RE) and easier, streamlined and less error prone modelling of the infrastructures available. Security was also in the focus of the project endeavours to make the framework more solid and more appealing to the end users. Figure 1 shows the updated high-level SODALITE General Architecture by layers.


© Copyright Beneficiaries of the SODALITE Project

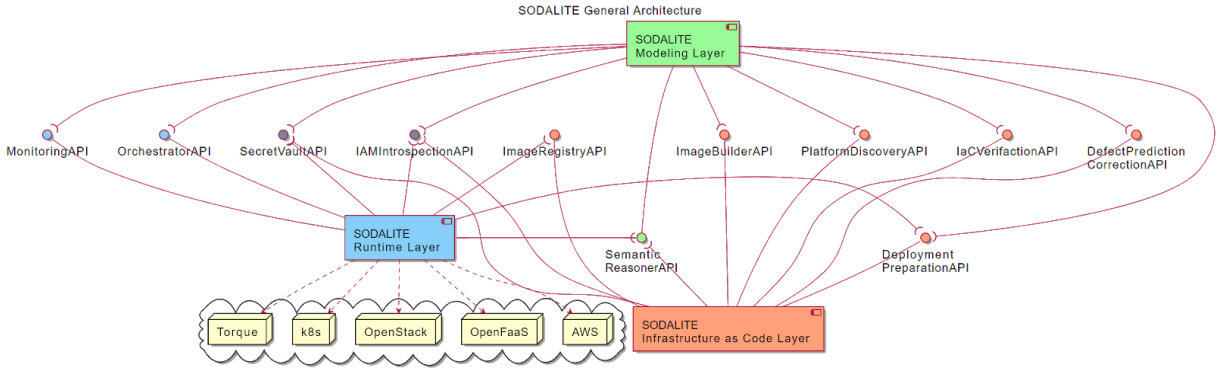


Figure 1 - Updated SODALITE layers general architecture

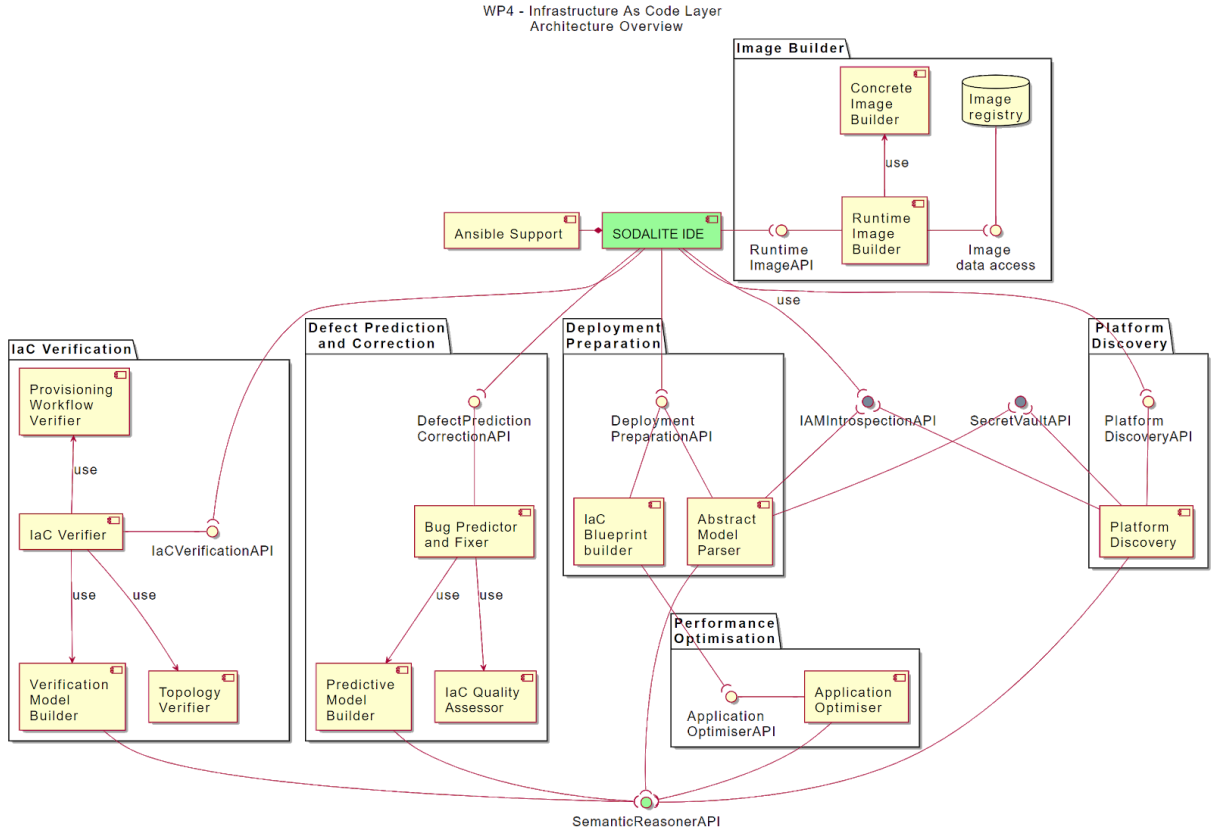


Figure 2 - Updated Infrastructure as Code Layer Architecture.

While there were limited changes in the SODALITE General Architecture, the IaC Management Layer shows some significant evolutions (see Figure 2). First of all, it has been integrated with the Security APIs introduced in the second year of the project and offering Identity and Access Management (IAM) for authentication and authorization of the requests and Secret Management API (Vault Service) for securely handling secret storage. Such APIs are now used by the Platform Discover and IaC Blueprint Builder component.

Second, the IaC Management Layer architecture shows two new components. These are the newly introduced Platform Discovery Service and the Ansible Support (Editor and Code Generator) that is developed within the SODALITE IDE.

# 3 IaC Layer development toolset and artifacts

In this section, we recall the rules adopted within SODALITE to produce and deliver code. Details on these aspects are available in [D2.4] and [D6.3]. Moreover, we provide an overview of the specific artifacts that are produced and made available to the users as part of the IaC Management Layer.


© Copyright Beneficiaries of the SODALITE Project

## 3.1 GitHub Repositories

SODALITE chose GitHub as its primary publicly available development version control system. GitHub is excellently managed for supporting the open source community, not only as a version control system, but also as a developers collaboration platform, by offering many available tools and further introducing concepts for this collaboration.

All the available open-sourced code produced in SODALITE can be reached through SODALITE's GitHub organization: https://github.com/SODALITE-EU.

SODALITE utilizes different project development collaboration features provided by GitHub such as: project boards, teams, issue tracker, pull requests, and peer reviews of code.

## 3.2 Continuous Integration and Continuous Delivery CI/CD

SODALITE uses the *Jenkins*[3] tool to support automated building, testing, versioning and publishing process for SODALITE components. To improve the quality and the automation of the CI/CD process a convention for the developed components has been set up with specific examples of usage described in detail deliverable D6.3 [D6.3].

## 3.3 Software QA

SODALITE is bound to produce mostly open source code on a publicly available version control system. The SODALITE consortium recognizes the high impact of developing excellent quality of code of its software components. For this reason the free and open online *SonarCloud*[4] utility is used to assess the quality of the code. To enable developers to deliver better code, SonarCloud shows various dashboards and enables a streamlined integration with GitHub, providing the developers with a good estimate of code quality even before merging the code into the master/main branch. This feature, among many others, is extensively used in the SODALITE CI/CD pipeline, providing both the developer and the reviewer of the code with significant and important insights about the quality of developed code, as well as providing useful suggestions on how to improve the code.

All of the repositories of the SODALITE components were integrated with SonarCloud during the second year of the project. The main metrics collected concern the following aspects:

- the number of bugs: bugs in SonarCloud are identified by exploiting various static analysis tools specific to the supported languages. [ref] and [ref] provide an overview of the types of bugs discovered by the tools used for Java and Python code, respectively,
- the number of security vulnerabilities and hotspots. As highlighted in the SonarCloud manual[5], "*with a Hotspot, a security-sensitive piece of code is highlighted, but the overall application security may not be impacted. It's up to the developer to review the code to determine whether or not a fix is needed to secure the code. With a vulnerability, a problem that impacts the application's security has been discovered that needs to be fixed immediately*",
- the number of code smells,
- the code coverage defined in terms of lines of code that are exercised by automated test cases,
- the amount of replicated code.

The general goal of SODALITE with respect to these metrics is to continuously keep them under control and improve them from release to release. As for the components belonging to the IaC

[3] https://www.jenkins.io/ The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.

[4] https://sonarcloud.io/ Code Quality assessment online tool Enhance - Your Workflow with Continuous Code Quality

[5] https://docs.sonarqube.org/latest/user-guide/security-hotspots/


© Copyright Beneficiaries of the SODALITE Project

Management Layer, we expect the metrics to show relatively high values considering that these components are used by most of the others in the platform.

In the following sections of this deliverable, for each stable component, we provide an overview of its current status in terms of the SonarCloud metrics.

### 3.4 IaC Management Layer artifacts

The following table provides links to the GitHub repositories, SonarCloud analysis reports and Dockerized images of each of the IaC Management Layer as it is shown in the architecture of Figure 2.

| Component | Github repository | SonarCloud dashboard | Dockerized image on Docker Hub |
|---|---|---|---|
| Image Builder | https://github.com/SODALITE-EU/image-builder | https://sonarcloud.io/dashboard?id=SODALITE-EU_image-builder | 1. https://hub.docker.com/r/sodaliteh2020/image-builder-cli<br>2. https://hub.docker.com/r/sodaliteh2020/image-builder-api<br>3. https://hub.docker.com/r/sodaliteh2020/image-builder-nginx<br>4. https://hub.docker.com/r/sodaliteh2020/image-builder-flask |
| Deployment Preparation (aka IaC Blueprint Builder) | https://github.com/SODALITE-EU/iac-blueprint-builder | https://sonarcloud.io/dashboard?id=SODALITE-EU_iac-blueprint-builder | https://hub.docker.com/r/sodaliteh2020/iac-blueprint-builder |
| IaC Verification | https://github.com/SODALITE-EU/verification | 1. https://sonarcloud.io/dashboard?id=SODALITE-EU_verification-unifiedapi<br>2. https://sonarcloud.io/dashboard?id=SODALITE-EU_verification-workflow<br>3. https://sonarcloud.io/dashboard?id=SODALITE-EU_verification-workflow | 1. https://hub.docker.com/r/sodaliteh2020/iacverifierapi<br>2. https://hub.docker.com/r/sodaliteh2020/workflowverifier<br>3. https://hub.docker.com/r/sodaliteh2020/toscasynverifier |
| Application Optimization (aka MODAK) | https://github.com/SODALITE-EU/application-optimisati | https://sonarcloud.io/dashboard?id=SODALITE-EU_applicat | https://hub.docker.com/r/modakopt/modak |


© Copyright Beneficiaries of the SODALITE Project

| | | | |
|---|---|---|---|
| | on/ | ion-optimisation | |
| Platform Discovery Service | https://github.com/SODALITE-EU/platform-discovery-service | https://sonarcloud.io/dashboard?id=SODALITE-EU_platform-discovery-service | https://hub.docker.com/r/sodaliteh2020/platform-discovery-service |
| Defect Prediction and Correction | https://github.com/SODALITE-EU/defect-prediction | 1. https://sonarcloud.io/dashboard?id=SODALITE-EU_tosca-smell<br>2. https://sonarcloud.io/dashboard?id=SODALITE-EU_ansible-defects<br>3. | 1. https://hub.docker.com/r/sodaliteh2020/ansiblesmells<br>2. https://hub.docker.com/r/sodaliteh2020/toscasmells |
| IaC Quality Framework | https://github.com/SODALITE-EU/iac-quality-framework | https://sonarcloud.io/dashboard?id=SODALITE-EU_iac-quality-framework | https://hub.docker.com/r/sodaliteh2020/iacmetrics |
| Ansible Abstract Playbooks Support | This is an IDE subcomponent https://github.com/SODALITE-EU/ide | https://sonarcloud.io/dashboard?id=SODALITE-EU_ide | https://hub.docker.com/r/sodaliteh2020/sodalite-ide |

# 4 New features developed in the second project year

## 4.1 Automated discovery and TOSCA description of infrastructure

Modelling different kinds of infrastructures into specific Resource Models (RM), that can then be referenced by the AADM, can be a tedious task for an IT Operation manager or a Resource Expert (RE). The knowledge needed to define a complex infrastructure spans over different areas and implies having comprehensive knowledge about the resource classification, capabilities, and dependencies and other platform specific aspects. This task usually requires extensive manual and iterative work for the transformation of the platform knowledge into the model definition, making it a hardly reachable goal for most of the RE. There are several objective reasons for this, with the most common ones being the specifics of the platform infrastructure topology and capabilities, which depend on an ever changing processes on most platforms, and the iterative manual work for abstraction of the resources defined in TOSCA. Consider a RE that wants to enable DevOps users (AOEs) to use resource models for different instances of Openstack (meaning many definitions of networks, storage types, flavors, image types etc.). Declaring all the resources through their node type definitions for an infrastructure can take days, and depending on the volatility of the Openstack instance, it should be regularly redefined, producing a consistent model of the resource


© Copyright Beneficiaries of the SODALITE Project

types available on the instance. Automating such a task provides an invaluable and tangible result for the RE, leaving him to define only specific aspects not handled by the automatic Platform Discovery Service and verify the correctness of the resource model.

SODALITE helps the Resource Expert (RE) to identify and create the platform RM definitions for typical HPC environments (managed by TORQUE[6]/SLURM[7] schedulers and accessible through SSH), Openstack private cloud environments and partial discovery of the AWS resources. The Platform Discovery Service creates standard TOSCA node type definitions that can be edited, verified and later injected in the SODALITE Knowledge Graph as part of the platform instance Resource Model (RM). These RMs are then used by the AOE within an AADM to support the modelling of a complex application deployment. The information supplied in the RMs describes the type of the resource, its workload capabilities (number of nodes, CPUs, GPUs or special hardware that might be used in the optimization process such as SSD drives attached etc.) or image types/flavors, volumes, networks, security rules for the VM management on Cloud infrastructure.

### 4.1.1 State Of The Art and Innovation

There are several papers considering approaches to use TOSCA, specifically to improve automated infrastructure model matchmaking including some work done in the field of automatic discovery of application services, but only one showing possible partial automatic discovery of infrastructure resources for specific infrastructures as described in this section.

[Wett 2016] presents an integrated modeling and runtime framework to enable the seamless and interoperable integration of different approaches to model and deploy application topologies. The framework is implemented by an open-source, end-to-end toolchain. Moreover, they validate and evaluate the presented approach to show its practical feasibility based on a detailed case study, in particular considering the performance of the transformation toward TOSCA.

[Tamburri 2019] articulates the foundations of the “intent modelling” approach, incorporating the most related modelling paradigm, that is, goal-modelling. They elaborate on it with a real, but simple industrial sample featuring the TOSCA language.

[Noudohouenou 2014] presents the Ubenchface tool, a framework for performance prediction and knowledge discovery. Inversely to traditional measurement methods and modeling, the proposed tool considers static metrics to analyze and tune application performance. This framework is more informative than simple benchmarking, or microbenchmarking. It is useful for performance investigations in similarity and redundancy study concerning benchmark suites, predicting, understanding scaling, and tuning.

Perhaps the most relevant paper is presented in [Brogi 2016]. Their work shows how the TOSCA standard can be exploited to provide a standard-based representation of the virtual machines and platforms offered by IaaS and PaaS cloud providers. DrACO is an open-source prototype tool that permits to look-up for cloud offerings and to retrieve them in a TOSCA format. The tool uses meta-indexes to gather information about accessibility, scalability and cost of VM offerings.

Nevertheless, using discovery for automatic platform TOSCA modelling is novel not only for Cloud platforms but especially for the HPC domains. SODALITE’s Platform Discovery Service uses cloud provider defined APIs and standard HPC cluster management tools to gather data about the

[6] https://adaptivecomputing.com/cherry-services/torque-resource-manager/ TORQUE is an industry-standard resource manager solution with higher adoption than any other resource management offering

[7] https://slurm.schedmd.com/overview.html Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters.


© Copyright Beneficiaries of the SODALITE Project

current state of the infrastructure and create a TOSCA model capturing the specifics of the platform resources.

**The SODALITE approach**

Platform Discovery Service will be able to output the description of the discovered platform in three different variants:

- TOSCA blueprint definition of the platform (can be reused in by any TOSCA orchestration supported framework) - supported in the released initial version,
- IDE DSL describing the discovered platform as a RM definition understandable by the SODALITE IDE (planned for M30)
- RM in the internal interchange turtle[8] format understandable by the semantic-reasoner and stored in the Knowledge base (planned for M30).

Having an automatically generated and consistent Resource Model definition introduces significant benefits for the Resource Expert:

- a much faster, more secure and less error prone way to create the Resource Model definitions,
- improves insights about the infrastructure resources usable through code intellisense, scoping and performance execution suggestions at design time,
- creates a platform for enabling the ad-hoc and runtime reconfiguration of the used resources based on the availability and capabilities of the resource.

Besides being innovative in its core by providing specific TOSCA platform definitions, Platform Discovery Service was designed to enable and provide a streamlined and multifunctional reuse of its key features and practices. Developed to adhere to code development design and best practices, it is still bound to the following three basic aspects.

**Open standards**

Tools used for platform descriptions support API design best practices and standards, namely OpenAPI and TOSCA.

**Flexibility**

Tools used for platform descriptions use template defined outputs and specific Ansible modules and collections. Each of the following points can be extended to provide maximum flexibility of the approach through:

- direct usage of SODALITE TOSCA blueprint and RMs generated (without any additional changes),
- extending existing Ansible collections for platforms (like in the case of Openstack security groups discovery),
- configuration and definition of different sets of templates for TOSCA or RM generation using standard jinja templating[9].

[8] https://en.wikipedia.org/wiki/Turtle_(syntax)

[9] https://jinja.palletsprojects.com/en/2.11.x/ Jinja is a modern and designer-friendly templating language for Python, modelled after Django's templates. It is fast, widely used and secure with the optional sandboxed template execution environment and extensively used in Ansible


© Copyright Beneficiaries of the SODALITE Project

**Reusability**

Using TOSCA Discovery Blueprints and Discovery Ansible collections separately or in a more integrated TOSCA definition provides the ability for platform discovery even without running Platform Discovery Service REST API.

### 4.1.2 Architecture

Platform Discovery Service consists of several modules, submodules and artifacts. At its core it uses xOpera[10], a lightweight orchestrator, as a library to execute TOSCA discovery blueprints for a predefined set of platforms. For each platform specialized Ansible collections or modules are used to implement specific resource discovery and gather resource description data in JSON format. The gathered data are then converted into TOSCA resource template definitions through the *jinja*[11] templating used in Ansible.

The TOSCA discovery blueprints used in the Platform Discovery Service represent another level of code reusability as they can be used separately with TOSCA orchestrators, such as xOpera, to create a TOSCA representation of the infrastructure even outside the Platform Discovery Service.

Platform Discovery Service (PDS) is implemented as a REST API with an API first - design driven methodology in place. The interface design is based on the OpenAPI 3.0 specification[12], a de-facto standard for developing REST API interfaces. OpenAPI Generator[13] toolset is used to create stubs for a Python REST API implementation, leading to a solid and design driven interface implementation.

Further the connexion[14] Python library is used to handle authentication described in the REST APIs and create a swagger UI for describing the API and adding the possibility to test and execute the REST API calls from a browser.

The API design driven process means easier maintainability and extensibility of the implementation with regards to changes in the underlying libraries used for business logic implementation. Any change in the underlying library is done through regeneration of the stubs and executing the tests again to ensure successful execution of regression tests.

Platform Discovery Service uses external services for IAM [Keycloak][15] to perform user authentication and authorization, as well as Secret Manager [Hashicorp Vault][16] for securely accessing and handling user secrets. This is needed in order to support authorization of the REST

[10] https://github.com/xlab-si/xopera-opera opera aims to be a lightweight orchestrator compliant with OASIS TOSCA. The current compliance is with the TOSCA Simple Profile in YAML v1.3

[11] https://jinja.palletsprojects.com/ Jinja is a modern and designer-friendly templating language for Python.

[12] https://www.openapis.org/ - The OpenAPI Initiative (OAI) was created by a consortium of forward-looking industry experts who recognize the immense value of standardizing on how APIs are described. As an open governance structure under the Linux Foundation.

[13] https://github.com/openapitools/openapi-generator/ OpenAPI Generator allows generation of API client libraries (SDK generation), server stubs, documentation and configuration automatically given an OpenAPI Spec (both 2.0 and 3.0 are supported).

[14] https://github.com/zalando/connexion connexion allows you to write an OpenAPI specification, then maps the endpoints to your Python functions; this makes it unique, as many tools generate the specification based on your Python code. You can describe your REST API in as much detail as you want; then Connexion guarantees that it will work as you specified.

[15] https://www.keycloak.org/ Open Source Identity and Access Management For Modern Applications and Services

[16] 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.


© Copyright Beneficiaries of the SODALITE Project

API call on behalf of the authorized user and to get access to the platform access tokens/keys or other credentials from the Secret Manager.

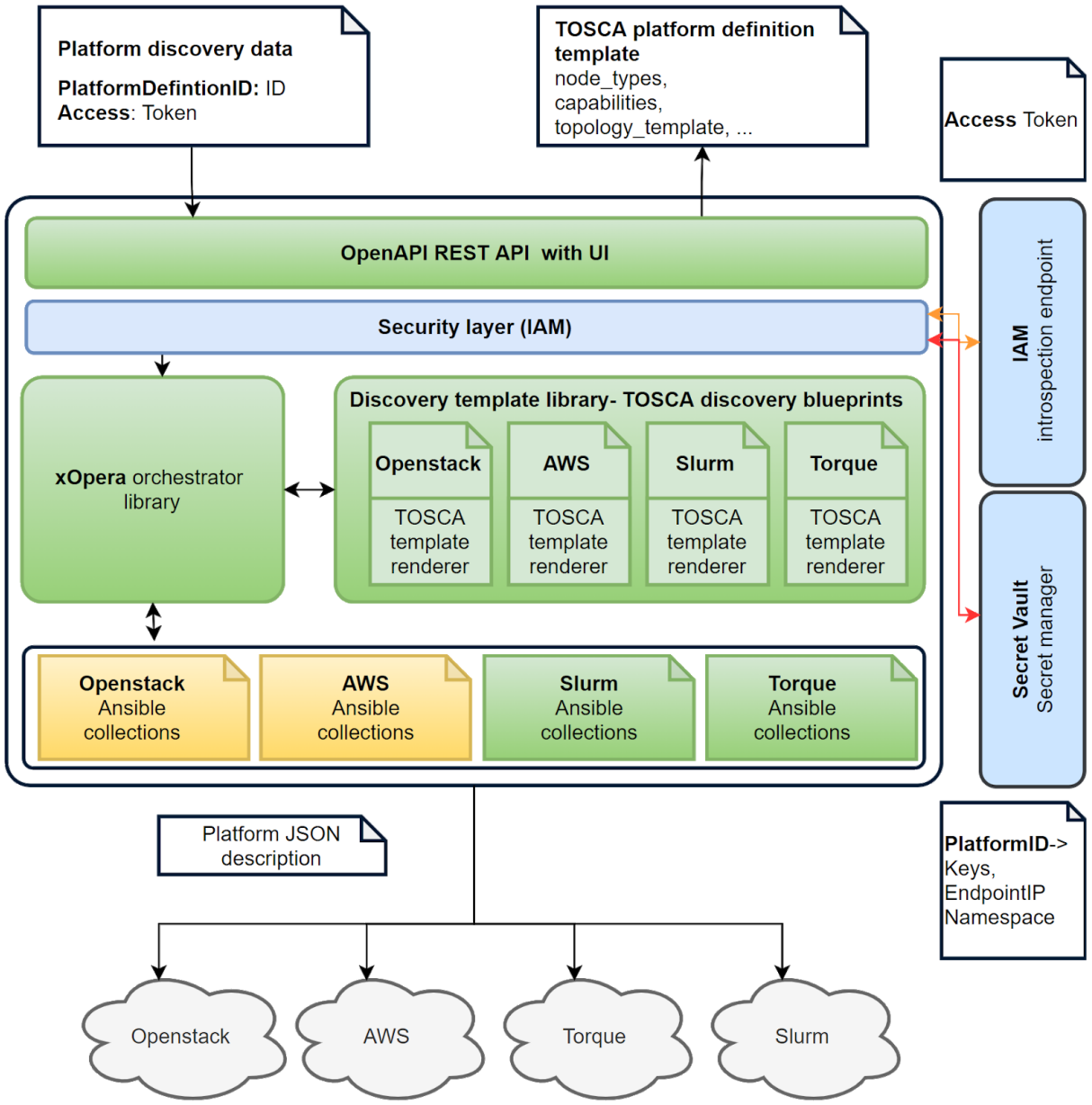


Figure 3 - Platform Discovery Service Architecture.

The multi-layered architecture of Platform Discovery Service is shown in Figure 3. Thanks to its modularity it leads to cleaner code, enhanced separation of concerns, improved maintainability and reusability.

### 4.1.3 Features

Following the multilayered architecture, the main features can be separated into three aspects:

- creation of definitions for platforms in using a TOSCA service template.
- creation of definitions for platforms using the semantic reasoner API to store the definitions in a common semantic exchange format.
- update of an existing definition of a platform using semantic reasoner API to store the updated definition in a common semantic exchange format.


© Copyright Beneficiaries of the SODALITE Project

Currently the first feature is implemented to the extent of representing different node capabilities respecting the **non-intrusive** resource discovery, meaning that, no additional nonstandard commands or tools are used to discover and describe the current state of the infrastructure. This is especially important for infrastructures or platforms that do not allow installation of system-wide tools usually with administrative (root) privileges like HPC clusters, at the same time keeping a lightweight process of resource discovery in place.

Steps describing the actual workflow for generating an automated TOSCA platform resource definition:

- get input and setup information about platform instance the service is accessing,
- authorize access and gather secrets such as access tokens, keys and discovery scope needed to access a specific platform instance on behalf of the user,
- generate a JSON description of a specific platform resource,
- generate a TOSCA node definition with regards to capabilities discovered for a specific platform resource,
- enable the Resource Expert (IT Ops) to apply specific changes to TOSCA node definitions before saving the resource models into the Knowledge base,
- enable Runtime integration with the Semantic reasoner and the Knowledge base to to supply AOE with fresh data and definitions also useful for runtime reconfiguration.

In the following short subsections the TOSCA Blueprint outputs produced by the Platform Discovery Service are presented, showing sample results of discovery for HPC Slurm, Openstack and AWS. Additionally tools and Ansible collections/modules used for the discovery are introduced.

**TORQUE**

For the discovery of the capabilities of a TORQUE managed cluster, the *pbsnodes* and *qstat* commands with its respective variants are used to describe the state of the cluster at a certain point in time. For this purpose a specialized Ansible collection has been designed and implemented covering the discovery of both *TORQUE* and *SLURM* managed HPC clusters.

**SLURM**

Discovering the capabilities of a SLURM managed cluster is enabled by the usage of standard SLURM *scontrol show node* and *scontrol show partition* commands with its respective variants to describe the state of the cluster at a certain point in time. A Platform Discovery Service generated TOSCA blueprint capturing this state is shown in Figure 5.


*© Copyright Beneficiaries of the SODALITE Project*

```
tosca_definitions_version: tosca_simple_yaml_1_3

capability_types:

  sodalite.capabilities.TestSlurm.Queue:
    derived_from: tosca.capabilities.Compute

  sodalite.capabilities.TestSlurm.WM:
    derived_from: tosca.capabilities.Compute

  sodalite.capabilities.TestSlurm.JobResources:
    derived_from: tosca.capabilities.Compute
    properties:
      gpus:
        type: integer
        required: true
      cpus:
        type: integer
        required: true
      memory:
        type: integer
        required: false
```

```
queue-hpc-TestSlurm-gpu:
  type: sodalite.nodes.hpc.TestSlurm.SlurmQueue
  properties:
    name: gpu
  capabilities:
    resources:
      gpus: 24
      cpus: 144
      memory: 1545600
  requirements:
    - wm: hpc-wm-slurm-TestSlurm-wm


hpc-wm-slurm-TestSlurm-wm:
  type: sodalite.nodes.hpc.TestSlurm.SlurmWM
  attributes:
    public_address: rmaister.hpc-rivr.um.si
    username: { get_input: user }
    ssh-key: { get_input: key-location }
  capabilities:
    resources:
      gpus: 24
      cpus: 9872
      memory: 40751720
```

Figure 4 - HPC SLURM TOSCA sample definition output.

**Openstack**

Openstack instance discovery is enabled by standard Openstack Ansible collections for getting information about the infrastructure through OpenstackAPI like:

- openstack.cloud.os_image_info[17] - for gathering information about Openstack images info for Operating Systems used created VM,
- openstack.cloud.os_flavor_info[18] - retrieves information about the flavors defined on the Openstack instance for creating a VM (VCPUs, Memory, Disk, etc),
- openstack.cloud.os_networks_info[19]- retrieves information about OpenStack networks.

Additional openstack simple collections were developed to discover security groups and keypairs registered with the Openstack instance:

- sodalite.discovery.os_security_group_info - retrieves Openstack security groups defined,
- sodalite.discovery.os_key_pair_info - retrieves list of SSH keys registered on Openstack accessible through the users account.

A part of the TOSCA blueprint covering the definition of an Openstack instance produced by the Platform Discovery Service is shown in Figure 4.

[17] https://docs.ansible.com/ansible/2.10/collections/openstack/cloud/os_image_info_module.html
[18] https://docs.ansible.com/ansible/2.10/collections/openstack/cloud/os_flavor_info_module.html
[19] https://docs.ansible.com/ansible/2.10/collections/openstack/cloud/os_networks_info_module.html


© Copyright Beneficiaries of the SODALITE Project

```
tosca_definitions_version: tosca_simple_yaml_1_3

data_types:
  sodalite.datatypes.OpenStack.TestOpenstack.SecurityRule:
    derived_from: tosca.datatypes.Root
    properties:
      protocol:
        required: True
        type: string
        default: tcp
        constraints:
          - valid_values: ['tcp', 'udp', 'icmp']
      port_range_min:
        required: True
        type: tosca.datatypes.network.PortDef
      port_range_max:
        type: tosca.datatypes.network.PortDef
        required: True
      remote_ip_prefix:
        default: 0.0.0.0/0
        required: True
        type: string
```

```
security-rules-node-exporter:
  type: sodalite.nodes.OpenStack.SecurityRules
  properties:
    ports:
     ports-tcp-9100-9100:
        port_range_max: 9100
        remote_ip_prefix: 0.0.0.0/0
        port_range_min: 9100
        protocol: tcp
    group_name: node-exporter
    group_description: node-exporter

network-xlab:
  type: sodalite.nodes.OpenStack.Network
  properties:
    name: xlab
    mtu: 1500

key-pair-alexander_maslennikov:
  type: sodalite.nodes.OpenStack.KeyPair
  properties:
    name: alexander_maslennikov
```

Figure 5 - Openstack TOSCA sample definition output created by Platform Discovery Service.

**AWS**

In the process of AWS platform discovery the standard *boto3*[20] Python library was used through standard AWS and community supported Ansible collections and modules:

- aws_region_info[21] - for getting information about the AWS supported regions,
- ec2_vpc_net_info[22] - for getting information about VPC (Virtual Private Cloud) - a logically isolated virtual network in the AWS cloud,
- ec2_vpc_subnet_info[23] - for getting information about VPC subnets defined,
- ec2_ami_info[24] - for getting information about available Amazon Machine Images (AMI) used to create a Virtual Machine (VM) in the Amazon Elastic Compute Cloud (EC2).

The sample Platform Discovery Service results in the form of a TOSCA blueprint for AWS after the executed TOSCA transformation, shown in Figure 6.

[20] https://github.com/boto/boto3 Boto3 is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python used in practically all Ansible collections and modules
[21] https://docs.ansible.com/ansible/latest/collections/community/aws/aws_region_facts_module.html
[22] https://docs.ansible.com/ansible/latest/collections/amazon/aws/ec2_vpc_net_info_module.html
[23] https://docs.ansible.com/ansible/latest/collections/amazon/aws/ec2_vpc_subnet_info_module.html
[24] https://docs.ansible.com/ansible/latest/collections/amazon/aws/ec2_ami_info_module.html


© Copyright Beneficiaries of the SODALITE Project

```
tosca_definitions_version: tosca_simple_yaml_1_3

capability_types:
  sodalite.capabilities.AWS.TestAWS.InstanceType:
    derived_from: tosca.capabilities.Root
    properties:
      name:
        type: string
        required: true
      vCPUs:
        type: integer
        required: true
      memory:
        type: integer
        required: true
      storage:
        type: integer
        required: true
      price:
        type: float
        required: true

topology_template:

  node_templates:
  #REGIONS
      sodalite-node-aws-region-TestAWS-eu-north-1:
      type: sodalite.nodes.AWS.TestAWS.Region
      properties:
        region_name: eu-north-1
        endpoint: ec2.eu-north-1.amazonaws.com

      sodalite-node-aws-region-TestAWS-ap-south-1:
      type: sodalite.nodes.AWS.TestAWS.Region
      properties:
        region_name: ap-south-1
        endpoint: ec2.ap-south-1.amazonaws.com


  #SUBNET
    sodalite-node-aws-subnet-TestAWS-subnet-61ad1a08:
    type: sodalite.nodes.AWS.TestAWS.VirtualPrivateCloud
    properties:
      subnet_id: subnet-61ad1a08
      availability_zone: eu-north-1a
      cidr_block: 172.31.16.0/20
      available_ip_address_count: 4091
    requirements:
      - region: sodalite-node-aws-region-TestAWS-eu-north-1
      - vpc: sodalite-node-aws-region-TestAWS-vpc-3257e15b

  #AMI
  sodalite-node-hpc-TestAWS-ami-72c34d0c:
    type: sodalite.nodes.AWS.TestAWS.AmazonMachineImage
    description:  Linux/UNIX \\
                  Canonical, Ubuntu, 18.10 Minimal, UNSUPPORTED daily amd64 cosmic minimal image built on 2018-10-16
    properties:
      image_id: ami-72c34d0c
      name: ubuntu-minimal/images-testing/hvm-ssd/ubuntu-cosmic-daily-amd64-minimal-20181016
      platform_details: Linux/UNIX
    requirements:
      - region: sodalite-node-aws-region-TestAWS-eu-north-1
```

Figure 6 - AWS TOSCA sample definition produced by Platform DIscovery Service.

The implementation of the Platform Discovery Service functionalities are left as open as possible enabling possible extensions on many levels:

- through the extension of the tools used for a specific platform discovery
- through the creation of additional Ansible collections targeting another aspect of the platform, by creating a JSON representation of the specific resource
- through TOSCA transformations of the JSON resource representation - e.g. using jinja templating for producing different TOSCA node definition output

Any user interested in reusing or extending the code can do so in various ways, such as:

- contributing directly to SODALITE Platform Discovery Service repository on GitHub,
- forking the repository and implementing its own changes,
- reusing the TOSCA discovery blueprints and Ansible collections for discovery in completely integrated way with a TOSCA orchestrator

### 4.1.4 Status

The current implementation is handling platform discovery and TOSCA definition creation for resources on OpenStack, HPC (TORQUE/SLURM) and partially AWS. For the discovery of HPC TORQUE and SLURM managed clusters a specialized Ansible collection has been developed. This collection will be further developed to ensure maximum usability. For other platforms, such as Openstack and AWS, specialized Python client API libraries were used in conjunction with respective Ansible collections. For Openstack additional modules for discovery of the security group definitions were developed, enabling upstream effort possible.

The Platform Discovery Service is analyzed by SonarCloud as shown in Figure 7. The results can be considered an excellent baseline for a component that is being released in its first version as a


© Copyright Beneficiaries of the SODALITE Project

research prototype. The quality can be further improved by removing two low priority hotspots and 19 codesmels in the next releases.

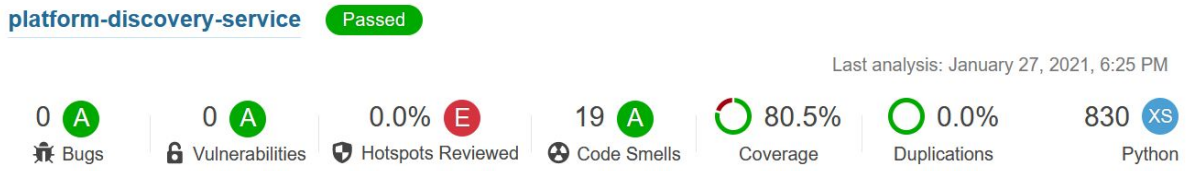


Figure 7 - PlatformDiscovery Service SonarCloud analysis

The code and the description of the component, its submodules and artifacts can be found in SODALITE github repository under https://github.com/SODALITE-EU/platform-discovery-service.

### 4.1.5 Next steps

At this point of development Platform Discovery Service still offers a number of possible improvements and extensions. Having defined the main features, upgrades will be implemented in the year 3 of the project to extend and enhance the integration into the SODALITE framework through automatic node definition updates using SODALITE semantic-reasoner API. One of the possible foreseen extensions is the Kubernetes [Kubernetes 2021] cluster node discovery.

The Platform Discovery Service can be further extended and improved by extending the discovery to other public clouds such as Google Cloud Platform and Azure.

## 4.2 Support to the creation of Ansible scripts integrated with the Resource Models

TOSCA blueprints alone are not enough to support the complete automation of deployment tasks. It allows to model what are the lifecycle operations of the components of a cloud application and their relationships, but it does not allow to model how these operations are implemented. This task is delegated to external scripting languages, one of the most prominent today being Ansible[25].

According to its documentation[26], *"Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates."*

Among the configuration management tools, one of the biggest points in favour of Ansible is the ease of set up, due to its agentless architecture. Thanks to it, Ansible allows to have only one master running on a server machine and able to configure nodes through SSH connections without the intervention of local agents.

The SODALITE main orchestrator, xOpera, complements TOSCA with Ansible scripts that are specific to the resource types used in the TOSCA code. As such, the TOSCA blueprint generated by the IDE starting from an AADM is not sufficient to accomplish the orchestration tasks and proper Ansible scripts must be defined and logically connected to the corresponding AADM and associated TOSCA blueprint.

The goal of this part of work is to extend the SODALITE approach to incorporate the generation of Ansible scripts as part of the resource modeling process. This, on the one side, makes the modeling process smoother and uniform as it does not force AOEs to edit Ansible files outside the SODALITE supporting tools and the IDE in particular. On the other side, it introduces, in the state of the art, a new and simple tool to support the development of a TOSCA-connected Ansible code which otherwise would have to be developed through a basic YAML editor.

The reader should note that the Ansible playbooks supporting the usage of the resources that are already supported by SODALITE have been already developed and are part of the SODALITE

[25] https://www.ansible.com/
[26] https://docs.ansible.com/ansible/latest/index.html


© Copyright Beneficiaries of the SODALITE Project

framework. This part of work aims at increasing the ability of Resource Experts to integrate new resources within SODALITE.

### 4.2.1 State of the art and innovation

The literature does offer some support to the development of Ansible playbooks. Visual Studio Code extension for Ansible[27] and Atom package[28] are two extensions for well-known software development editors that support the development of Ansible scripts. They offer auto completion support and syntax highlighting. With the Visual Studio Code extension, the user is also helped in choosing the module to use, as the content assistant shows the list of all the available modules. A list of the parameters that can be passed to each module is also provided. They, however, do not support the writing of a playbook in the context of a TOSCA operation, which passes some inputs to it and uses it for the implementation of the operation itself. As such, their usage in the context of the development of a TOSCA-based orchestration would require the developer to manually create such correspondence and check the coherency of naming conventions and parameters.

An interesting approach that, like SODALITE, adopts a model-driven approach is CloudCAMP [Bhattacharjee 2016, Bhattacharjee 2017]. It focuses on transforming a provided business model into deployable artifacts and on providing to the user a DSML for abstracting the design requirements. It also generates Ansible scripts based on the requirements provided by the user and by gathering predefined templates from a database. A similar approach for what concerns the generation of Ansible playbooks is offered by UPSARA, which otherwise is focused on modelling performance analysis experiments [Barve 2018].

Both approaches do not really offer the possibility to the user to model directly the Ansible playbook through a DSL, which is instead a feature that SODALITE aims at providing, in order to give to users (Resource Experts in this case) a significant degree of freedom in terms of the actions that can be encoded in an Ansible playbook.

In our approach we have extended the SODALITE IDE to include support for the creation of *abstract Playbooks* explicitly associated with the resource types defined in a Resource Model. Such abstract playbooks are then translated into concrete Ansible scripts that can be executed by xOpera.

The main characteristics that make our approach novel with respect to the state of the art are the following:

- Guided creation of the abstract playbooks following the general IDE approach. As discussed before, to the best of our knowledge, our IDE extension is the first one that offers TOSCA-specific guidance in the creation of Ansible scripts.
- First complete metamodel for Ansible and systematization of all its concepts.
- Coherent organization of attributes within an Ansible entity.
- Coherent integration of the abstract playbooks with the corresponding resource types defined in the resource models.

In the following section these features are described in further details.

### 4.2.2 Features

#### Guided creation of the abstract playbooks

The SODALITE IDE has been extended to support guided editing of Ansible abstract playbooks. The syntax offered by the IDE, which we call Ansible DSL, generalizes the concepts defined by the Ansible language and integrates them with resource models.

Thanks to the checks introduced by the editor, the abstract playbooks are, by construction, coherent with the resource types to which they are associated. They are then translated into

[27] https://marketplace.visualstudio.com/items?itemName=vscoss.vscode-ansible
[28] https://atom.io/packages/autocomplete-ansible


© Copyright Beneficiaries of the SODALITE Project

Ansible playbooks that can be executed by the Ansible executor and used by the xOpera orchestrator.

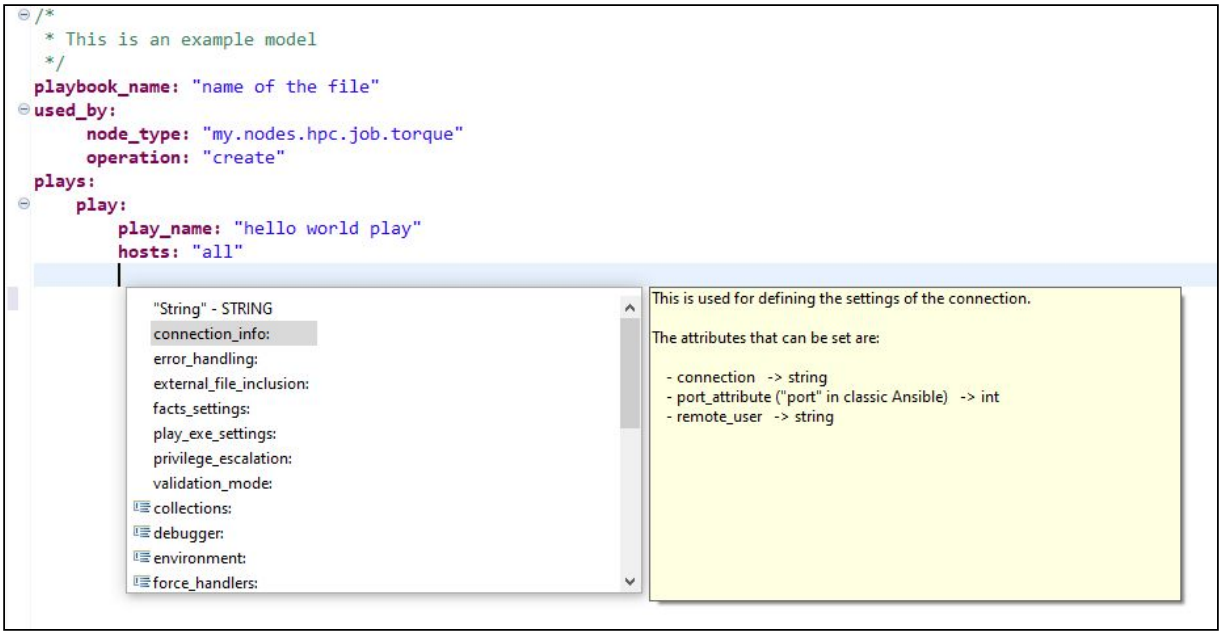


Figure 8 - Ansible DSL editor.

Figure 8 shows a portion of the editor UI highlighting the content assistance mechanism. In the example, the content assistance suggests the user the possibility to add to the playbook three possible attributes concerning connection-specific information, “connection” and “remote_user” of type string and “port_attribute” of type integer. In the DSL, Ansible attributes are grouped in categories depending on their usage. Figure 8 highlights the categories we have defined and shows that the three attributes mentioned above are part of the connection_info category. We expect that the organization of the attributes will simplify their usage as part of the coding activity.

**First complete metamodel for Ansible and systematization of all its concepts**

The integration of Ansible in SODALITE has started from the creation of a metamodel that describes the main elements of the language and defines the integration points with the other elements of an AADM. The main source used for understanding Ansible in order to build the metamodel was its documentation[29] together with the TOSCA standard [Rutkowski 2020].

The metamodel is available in two versions, a conceptual version which is useful to convey the main Ansible elements and the way it relates to TOSCA node types and an implementation version which completely describes the Ansible DSL we have defined. In this section we present the conceptual metamodel. The implementation metamodel is available as an appendix to this deliverable.

Figure 9 presents the main concepts of Ansible. A playbook, which implements a TOSCA interface parameter, can have multiple roles and is composed of one or more plays. These include executions and variables that can be of three different types. One of these is input which is correlated to the pieces of information that are received through a TOSCA interface parameter. nodeType, nodeTypeDescription and interface derive directly from the TOSCA metamodel.

[29] https://docs.ansible.com/ansible/latest/index.html.


© Copyright Beneficiaries of the SODALITE Project

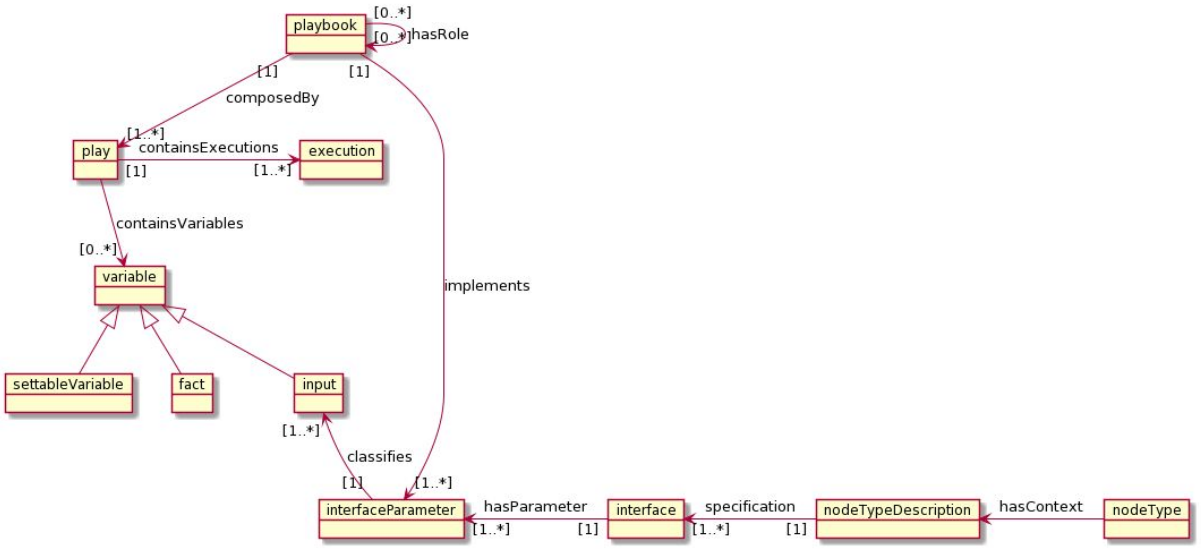


Figure 9 - Ansible metamodel - general view.

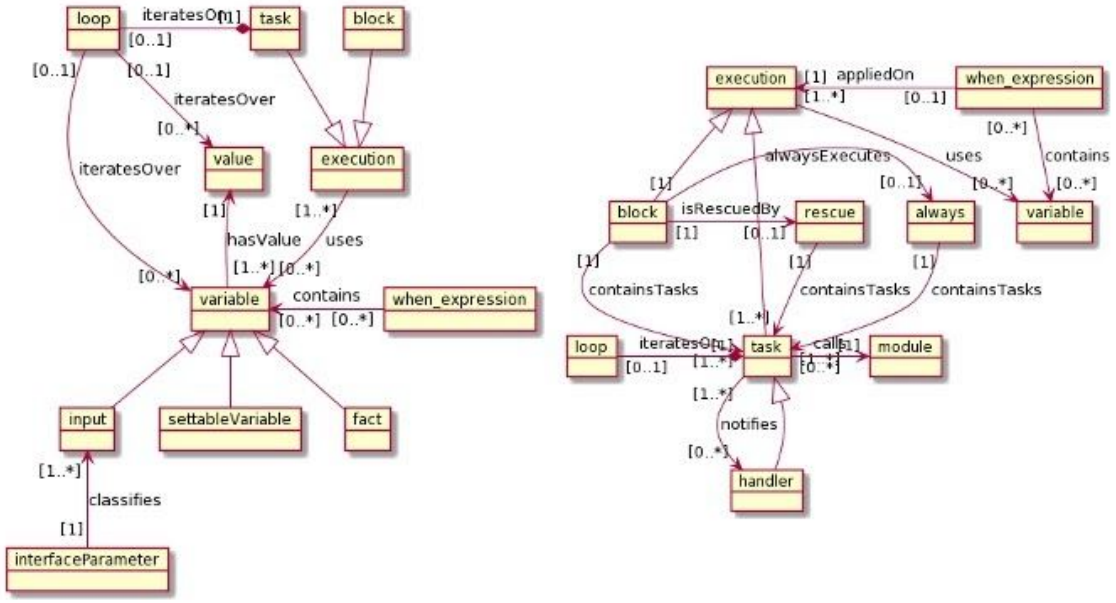


Figure 10 - Ansible metamodel - focus on variable and execution.

Figure 10 highlights the specialization of execution in task and block, the possibility to define, as part of a block, loops that iterate on the values of a variable, as well as the possibility to contain other tasks. Variables can contain “when expressions” that, when verified, determine the execution of a block or a task. Tasks can call modules made available on the Ansible repository. Each of these focuses on the execution of specific actions.

### 4.2.3 Architecture

As mentioned in the previous section, the support to the creation of Ansible scripts is developed as part of the IDE. More specifically, a new Eclipse plugin has been created that relies on the Resource


© Copyright Beneficiaries of the SODALITE Project

Model plugin. The new plugin is implemented using Xtext[30]. Thanks to this last framework, the implementation consisted in the following steps:

- The creation of the abstract playbooks grammar based on the implementation metamodel presented in the Appendix.
- The execution of the process automated by xtext that leads to the creation of the software structure and to the generation of the data structures needed to represent the abstract syntax tree derived from the grammar.
- The customization of the wizard to support the users in the creation of an Ansible project
- The development of the content assistance.
- The development of the Ansible YAML generation logic.

### 4.2.4 Status

The first version of the editor and generation mechanism is available as part of the IDE in its same repository.

More specifically, the github folders that include the Ansible-specific parts are all positioned under the following folder: https://github.com/SODALITE-EU/ide/tree/ansible/dsl/org.sodalite.IDE.parent and include the following elements:

- *"org.sodalite.dsl.ansible"*[31]: this folder includes the core of the system, namely, the defined grammar, the data structures automatically generated by xtext based on the grammar, the Ansible generator, and the scoping mechanism.
- *"org.sodalite.dsl.ansible.ui"*[32]: this folder includes the part concerning the integration with the Eclipse UI. In particular,the content assistant and the Eclipse wizard for creating an Ansible DSL project with a starting template.
- *"org.sodalite.dsl.ansible.ui.tests"*[33]: this folder is dedicated to testing the components in the "org.sodalite.dsl.ansible.ui" folder.
- *"org.sodalite.dsl.ansible.tests"*[34]: this folder is dedicated to testing the components in the *"org.sodalite.dsl.ansible"* folder.
- *"org.sodalite.dsl.ansible.ide"*[35]: this folder is dedicated to functionalities of the IDE that are platform-independent.

Being part of the IDE, it is not possible to provide information about code quality that is independent from what is available for the whole component. Such information is available here https://sonarcloud.io/dashboard?id=SODALITE-EU_ide.

### 4.2.5 Next steps

To complete the support to the creation of Ansible playbook, we intend to integrate as part of the IDE also the possibility to search through the Ansible Modules available online and to select the right one to be used within an abstract playbook.

Another possible task that might be considered, if possible, given the project resource constraints and other commitments concerns, is the possibility to develop a backward transformation from Ansible playbooks to the corresponding abstract playbooks handled by the Ansible editing support. This feature could be useful to simplify the modification of preexisting playbooks within the context of the SODALITE framework.

[30] https://www.eclipse.org/Xtext/
[31] https://github.com/SODALITE-EU/ide/tree/ansible/dsl/org.sodalite.IDE.parent/org.sodalite.dsl.ansible
[32] https://github.com/SODALITE-EU/ide/tree/ansible/dsl/org.sodalite.IDE.parent/org.sodalite.dsl.ansible.ui
[33] https://github.com/SODALITE-EU/ide/tree/ansible/dsl/org.sodalite.IDE.parent/org.sodalite.dsl.ansible.ui.tests
[34] https://github.com/SODALITE-EU/ide/tree/ansible/dsl/org.sodalite.IDE.parent/org.sodalite.dsl.ansible.tests
[35] https://github.com/SODALITE-EU/ide/tree/ansible/dsl/org.sodalite.IDE.parent/org.sodalite.dsl.ansible.ide


© Copyright Beneficiaries of the SODALITE Project

## 4.3 Optimization - MODAK

Software application developers and users are now targeting a wide range of diverse computing platforms, such as on-premise supercomputers and clouds with heterogeneous node architectures. Compute intensive applications like High Performance Computing (HPC) or Artificial Intelligence (AI) training also have requirements of specialised execution environments, including computing accelerators, high speed interconnects, faster memory, and storage. Even if software-defined environments provide both flexibility and portability, we still need applications to optimally use and benefit from these diverse resources.

The MODAK package, a software-defined optimisation framework for containerised HPC and AI applications, is the SODALITE component responsible for enabling the static optimisation of applications before deployment. MODAK aims to optimise the performance of application deployment to infrastructure in a software-defined way. Automation in application optimisation is enabled using performance modelling and container technology. Containers provide an optimised runtime for application deployment based on the target hardware and along with any software dependencies and libraries.

### 4.3.1 Innovation

The convergence of the cloud and HPC has made the deployment and management of applications on these heterogeneous infrastructures paramount. Container virtualization has grown in popularity as a bridge between these heterogeneous environments due to the ease of use, portability, scalability, and the advancement of user-friendly runtimes. It poses a simple way to share scientific applications and reproduce research on either cloud or HPC systems. This has driven the deployment of scientific HPC and AI applications on cloud infrastructure that offers alternative cost models, as well as the convergence of cloud methodologies with traditional HPC systems used to improve the user experience. A number of tools use containers to optimise application deployments. ConfAdvisor [Chiba 2019] is a tuning framework for containers on Kubernetes. AWS compute optimiser optimises workloads for both cost and performance based on historical utilization metrics [AWS 2020]. Other works have proven that MPI containers can be deployed on HPC using Docker [de Bayser 2017]. In this respect, MODAK will provide a common way to run on both cloud and HPC systems.

While most AI applications can be deployed in containers, this is not the default option for HPC applications. With diverse hardware and software dependencies and libraries, building or selecting an optimised container for application deployment is crucial. For example, MPI libraries on the host machine and in the container must match when deploying HPC applications in order for the container to use the hardware-optimised version of MPI available on the host. AI training frameworks require target-specific libraries and drivers to be configured. Even though Docker and Singularity support labelling of containers, they are seldom used when developing them.

To overcome this issue, MODAK maps the optimal application parameters to the infrastructure target by building or selecting an optimised container and then encoding optimisations in a job script. This is accomplished by abstracting performance requirements in an optimisation Domain Specific Language (DSL) and enabling performance prediction through application and infrastructure performance models. MODAK can also auto-tune and auto-scale applications based on user-created optimisation models. MODAK additionally supports batch schedulers like SLURM and TORQUE to provide ease of use in running HPC jobs in supercomputers and cloud.

Other tools provide similar MODAK features. The HPC Container Maker framework [McMillan 2018] provides functions to configure applications and dependencies for building container images in Docker and Singularity formats. However, it does not provide a way to deploy on batch systems. EASEY [Höb 2020] enables not only building application containers for target clusters and MPI


© Copyright Beneficiaries of the SODALITE Project

libraries, but also manages the deployment, job management, and data staging. While this approach is similar to that of MODAK, it does not model the performance optimisations.

### 4.3.2 Architecture

Figure 11 gives an overview of the MODAK components. We start with a high-level application API for the three types of applications supported: AI training and inference, traditional HPC (MPI and OpenMP parallelised), and big data analytics. We pass this information to MODAK, which matches it with the performance model outputs to produce a job script for the execution submission and an optimised container. MODAK can also auto-tune and auto-scale applications based on user input.

In summary, MODAK requires the following inputs:

- Job submission options for batch schedulers such as SLURM and TORQUE
- Application configuration such as application name, run and build commands
- Optimisation DSL with the specification of the target hardware, software libraries, and optimisations to encode. Also contains inputs for auto-tuning and auto-scaling.

Then, MODAK produces a job script (for batch submission) and an optimised container that can be used for application deployment. An image registry contains MODAK optimised containers while performance models, optimisation rules and constraints are stored and retrieved from the Model repository. Singularity container technology was chosen to provide a portable and reproducible runtime for the application deployment, due to better performance and native support for HPC.

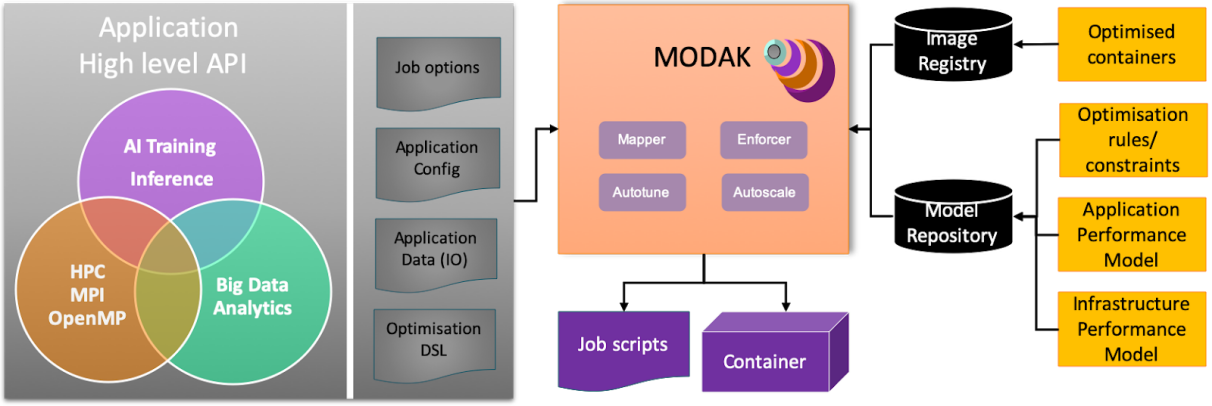


Figure 11 - MODAK architecture.

### 4.3.3 Features

MODAK automates optimisation using four main components, as described below:

- **Mapper**

  The Mapper maps application deployment to an optimised container based on the user specified input (DSL) and labels for containers. Containers provide an optimised runtime for application deployment based on the target hardware and along with any software dependencies and libraries. MODAK labels the containers based on optimisation support for diverse hardware and software, and then uses these labels to map the optimised containers to application deployments. Figure 12 shows a DSL example that labels the MPICH container (i.e. a container that includes the MPICH distribution) with the application name, version, and support for hardware like x86, NVIDIA GPUs, and software such as specific compilation commands for MPI applications.


© Copyright Beneficiaries of the SODALITE Project

```
"name": "mpich_container",
"app_name": "mpich",
"version": "2.2",
"hardware_support": {
  "enable_opt_build": true,
  "cpu_type": "x86",
  "acc_type": "nvidia"},
"software_support": {
  "mpic++": "true",
  "mpicc": "true",
  "mpifort": "true"}
```

Figure 12 - Container mapping DSL.

- **Enforcer**
  The optimisation process depends not only on application and infrastructure but also on the configuration and data. MODAK allows users to define optimisation rules that are enforced for deployment. The Enforcer component returns the optimisation script to be used based on the rules and user-selected optimisations in the input DSL. For example, enabling graph compiler-based optimisations in an AI framework requires environment settings to be modified. For MPI-based applications, there are many environment settings that change the way message passing is optimised based on message size and communication pattern. Data-related optimisations may involve the possibility to automatically copy the data to fast disks, if available, to improve I/O bound applications. MODAK can embed the chosen optimisations in the job script submitted to a batch scheduler.
- **Tuner**
  Autotuning enables users to automatically search possible application deployments for the desired result. MODAK's Tuner supports the CRESTA autotuning framework [Schliephake 2012.] The framework defines a DSL to expose the tuning choices as parameters, constrain and inject them into the application source, then build and run the application. The framework supports an exhaustive search of the parameter space and can tune for any metric output, not just runtime.
- **Scaler**
  In MODAK, we can predict the efficiency and speedup of an application on N nodes based on the performance prediction model. We used a combination of benchmarking to model infrastructure and then analytical modelling to model application runtime (see [D3.3]). This allows MODAK to automatically scale applications to certain numbers of nodes based on the model prediction. Using the parallel efficiency metric specified by the user in the optimisation DSL, the Scaler aims to predict the scale at which parallel efficiency is achieved and automatically increase the number of nodes of the deployment (Autoscale). The Scaler can also enable or disable accelerator, memory, and storage devices based on the model and availability in the target.

### 4.3.4 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 (see [D6.3] and [D6.6]). Currently, MODAK supports TensorFlow, PyTorch, MXnet, mpich, and openmpi containers for x86 and NVIDIA GPUs. These containers are further labelled with version requirements and support for


© Copyright Beneficiaries of the SODALITE Project

optimisations like graph compilers or BLAS/LAPACK. A preliminary MODAK integration in the SODALITE framework has been implemented (see [D6.3]). The Tuner and Scaler features are not yet implemented. MODAK repository is available here https://github.com/SODALITE-EU/application-optimisation and is already integrated with SonarCloud[36]. The quality values are being continuously improved and indicate at the moment the presence of some code smells that will be addressed in the next months.

#### 4.3.5 Next steps

MODAK will be extended during the third year of the project as follows:

- Complete integration into the SODALITE framework
- Include data-related optimizations to improve I/O performance. This includes the possibility to automatically copy or cache data on I/O fast devices before the application starts its execution. MODAK will require the knowledge of the available storage resources (filesystem type and paths) and will use some heuristics to check if the operation is convenient
- Include the Tuner and Scaler features, and extend the performance model to include GPU execution
- Add support of cloud systems
- Add support for big data analytics applications.

### 4.4 Analytics and Semantic Support

Infrastructure as code (IaC) simplifies the provision and configuration of the IT infrastructure at scale. As the size and complexity of IaC projects increase, it is critical to maintain the code and design quality of IaC Scripts. According to a recent report on Cloud Threat[37], nearly 200,000 insecure IaC templates were found among IaC scripts used by a set of enterprises, and 65% of cloud incidents are due to misconfigurations. Thus, the detection and correction of defective and erroneous IaC scripts is of paramount importance. To this end, we systematically identify and classify the common types of smells and bugs in IaC (Section 4.4.1). A software smell is any characteristic in the artifacts of the software that possibly indicates a deeper problem or quality issue [Sharma 2018]. We also develop the tools that can detect such smells/bugs and potentially suggest fixes (Section 4.4.2).

#### 4.4.1 IaC Taxonomies

##### 4.4.1.1 Innovation

We have created three novel taxonomies/catalogs for IaC, covering best and bad practices, smells, and bugs. While there exist recent works on bug and smell catalogs for IaC, they exhibit several limitations. Most catalogs focus on a subset of bug types or smell types, for example, security smells, and implementation smells for a subset of IaC language constructs. Moreover, the smell taxonomies use the best and bad practices that were extracted from one or a few sources. Finally, there are no unified taxonomies across different (widely-used) IaC languages such as Ansible, Puppet, and Chef. Thus, in SODALITE, we first develop a taxonomy of best and bad practices in three IaC industrial languages (Ansible, Puppet, and Chef), based on an analysis and synthesis of the multi-vocal literature. Based on this best/bad practices taxonomy, we develop an unified smell taxonomy for IaC. In addition, we create a bug taxonomy, based on a qualitative analysis of bug fix related commits collected from open source software repositories. We also adopt these taxonomies for TOSCA.

[36] https://sonarcloud.io/dashboard?id=SODALITE-EU_application-optimisation
[37] https://start.paloaltonetworks.com/unit-42-cloud-threat-report


© Copyright Beneficiaries of the SODALITE Project

#### 4.4.1.2 Methodology

- **IaC Best and Bad Practices Catalog**

  We investigated infrastructure code language/tools and best/bad practices from a practitioner perspective by addressing grey literature in the field, stemming from 65 selected sources and systematically applying qualitative data analysis techniques. We build our research methodology upon the guidelines proposed in systematic literature reviews in software engineering [Kitchenham 2009]. Figure 13 shows the steps of our methodology. We first define the goals of the grey literature review, formulate the research questions to address those goals, and then using Google search engine, the literature sources are searched. By applying a set of exclusion and inclusion criteria and quality assessment criteria, a filtered set of literature sources are selected. The selected sources are systematically coded using descriptive coding techniques [Saldaña 2015] . The codes are extracted, analyzed, grouped, and synthesized to identify the key best and bad practices reported by the IaC practitioners.

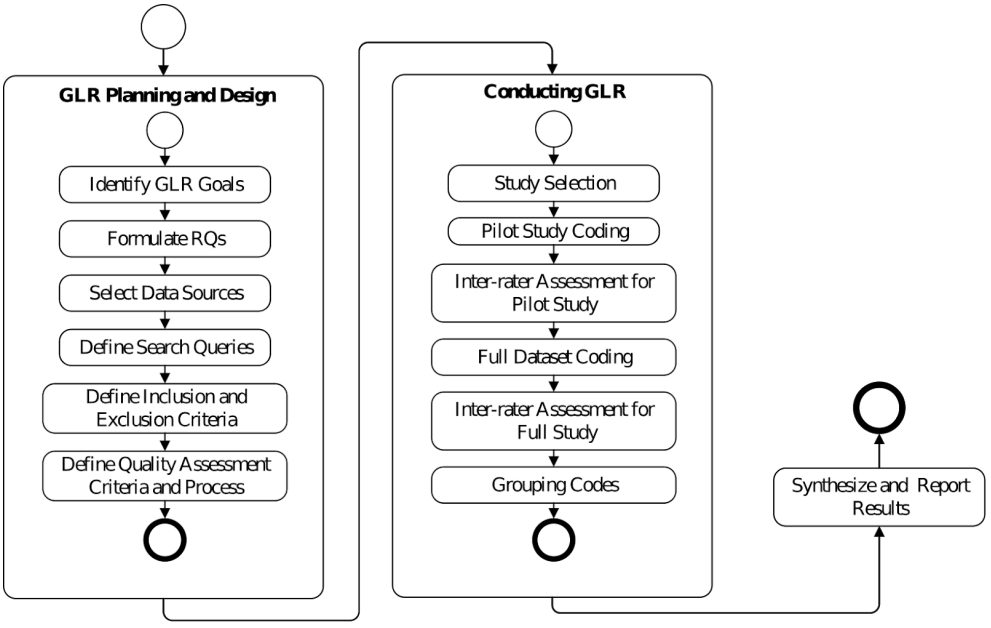


Figure 13 - An overview of the methodology for creating best/bad practices taxonomy.

- **IaC Smell Catalog**

  Figure 14 shows the overview of the methodology for creating IaC smell taxonomy. As in the existing literature, we define the smells as the violations of IaC best practices and the use/occurrence of IaC bad practices. Thus, our best and bad practices taxonomy is a key input to the IaC taxonomy. In the software smell literature, different types of smells have been reported for different types of programming languages and systems. The smells for IaC languages should be aligned with those smell categories reported in the literature. Thus, based on a set of recent systematic literature reviews on software smells, we first find a set of candidate smells types. Next, we map the violation of the IaC best practices or application of IaC bad practices to a subset of candidate smells types, which results in a candidate taxonomy for IaC smells. In order to validate the compiled taxonomy, we conduct a survey with the IaC developers or practitioners. Based on the feedback from the developers, we refine and update the taxonomy.


© Copyright Beneficiaries of the SODALITE Project

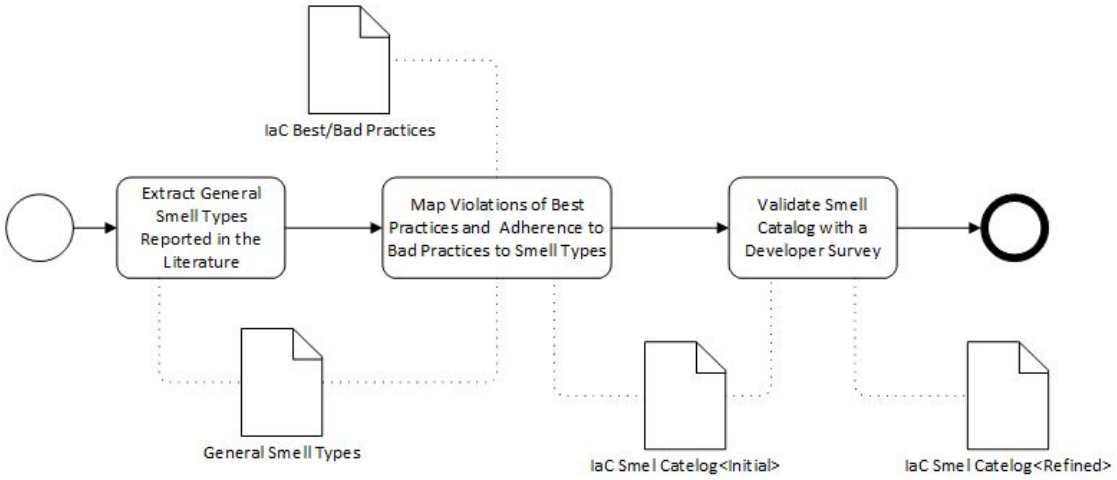


Figure 14 - An overview of the methodology for creating the IaC smell taxonomy.

- **IaC Bug Catalog**

  Figure 15 shows the overview of the methodology for creating the IaC Bug taxonomy. A common approach to create a bug catalog is to analyze bug fix commits. Thus, we first systematically selected a set of open source IaC (Ansible) repositories and then mined the bug fix commit messages. We used the criteria and guidelines used by the existing research for both selecting repositories and mining bug fix commits. By applying descriptive coding to the information in the commit messages, the bug categories are identified and the identified categories are mapped to those in the existing bug taxonomies. As regards to the existing bug taxonomies, we use a recent taxonomy for IaC (Puppet IaC language) [Rahman2 2020]. In addition, Common Weakness Enumeration (CWE)[38] is also used. CWE is widely used in bug taxonomy research literature. The created bug taxonomy is validated through a survey with the IaC developers. Finally, based on the feedback from the developers, we refine and update the taxonomy.

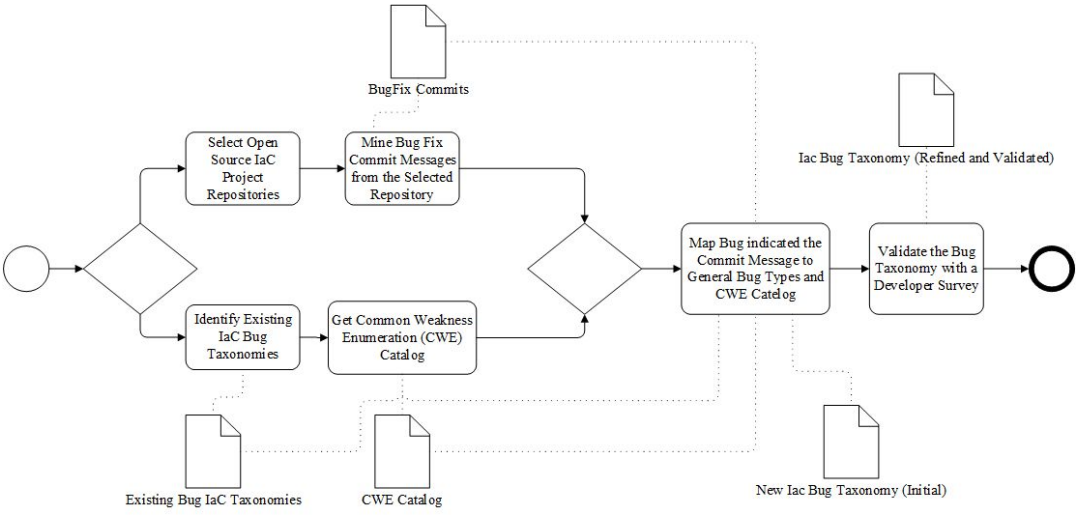


Figure 15 - An overview of the methodology for creating the IaC bug taxonomy.

[38] https://cwe.mitre.org/


© Copyright Beneficiaries of the SODALITE Project

#### 4.4.1.3 Features

The IaC taxonomies provide the complete lists of best and bad practices related to the development of IaC scripts, and the common types of smells and bugs that occur in IaC scripts. Smell and bug taxonomies are validated with the surveys with IaC developers/practitioners. The best and bad practices taxonomy is created based on the literature produced by the IaC developers/practitioners.

#### 4.4.1.4 Status

- **IaC Best and Bad Practices Catalog**

  We identified 224 IaC best practices: 20 language-agnostic, 39 for Ansible, 70 for Chef, and 88 for Puppet. The practices cover each of the key constructs/abstractions of IaC languages. They reflect both implementation issues (e.g., naming convention, style, formatting, and indentation) and design issues (e.g., design modularity, reusability, and customizability of the code units of the different languages).

  We identified 54 IaC bad practices in total: 13 language-agnostic, 14 for Ansible, 8 for Chef, and 19 for Puppet. While most of these practices concern design and implementation issues related to key constructs/abstractions of IaC languages, they also reflect the violations of the essential principles of IaC: idempotence of configuration code, separation of configuration code from configuration data, and infrastructure/configuration management as software development.

  A journal publication is under review at Information and Software Technology journal (the second revision).

- **IaC Smell Catalog**

  The use of the bad practices as well as the deviations from (or violations of) the best practices are defined as anti-patterns/smells. Based on the identified 224 IaC best practices and 54 IaC bad practices, we have created a IaC smell taxonomy of 45 high-level smell types.

- **IaC Bug Catalog**

  We mined 4493 bug fix commits from 26 open source Ansible repositories. Then, we further filtered those commits and identified 1071 commits as the final dataset. Next, we mapped those into the existing bug taxonomy for Puppet IaC. Among 1071 commits, 471 commits were not mapped to any of the categories in that taxonomy. Thus, we mapped those 471 commits into the bug categories proposed in CWE (Common Weakness Enumeration), which resulted in 6 new bug categories.

#### 4.4.1.5 Next steps

- IaC Smell Catalog

  We have planned a survey with IaC developers for validating the created IaC taxonomy. Once the results of the developer survey are received, we plan to write a journal paper.

- IaC Bug Catalog

  First, we will run a survey with the IaC developers for validating the bug taxonomy. Then, when the results of the developer survey are ready, we will write a journal paper.


© Copyright Beneficiaries of the SODALITE Project

### 4.4.2 IaC Defect Prediction and Correction

#### 4.4.2.1 Innovation

In software engineering literature, data-driven models (e.g., machine learning) and rule-based models have been used to detect smells and bugs in the source code of different programming languages. Recently, the software engineering community has paid attention to bug and smell detection in IaC. The rule-based techniques have been also applied to detect defects in infrastructural code scripts such as Puppet and Chef Scripts, e.g., security smells in Puppet [Rahman1 2019], implementation and design smells in Puppet [Sharma 2016] and implementation and design smells in Chef [Schwarz 2018]. Most industrial IaC smell detectors (i.e., so-called Linter tools), for example, Ansible Lint[39] and Puppet Lint[40], also use a rule-based approach. However, none of these works use semantic models of IaC and semantic rules or aim to unify the smell detection across different IaC languages. Moreover, the different types of smells/bugs may potentially need different techniques for their detection [Sharma 2018].

- **Semantic Formal Rule-based Smell Detection.**
  Compared with the existing studies, SODALITE proposes a semantic rule-based approach to detect the smells and antipatterns in IaC, for example, smells in TOSCA blueprints and Ansible scripts. Our framework facilitates the generation of knowledge graphs to capture TOSCA-based deployment models. The aim is to map IaC codes to self-contained, independent and reusable knowledge components, amenable to analysis and validation using Semantic Web standards, such as SPARQL. To explain detected smells and recommend fixes, the initial semantic models are extended to specify smells, their causes, and their fixes. A semantic approach helps us to deal with structure and semantic relations over resources, their relationships and the properties. The semantic reasoning process is able to draw new and hidden knowledge from the existing information. It could also enable us to build a unified framework to detect smells across different IaC languages by utilizing semantic Web techniques such as ontology alignment and query rewriting.

- **Deep Learning and NLP for Detecting Linguistic Anti-patterns and Misconfigurations**.
  There is an emerging trend in software defect prediction for using deep learning and natural language processing (NLP), in particular, code embeddings (code vectors) [Alon 2019][Liu 2019]. However, there are no similar studies on IaC defect prediction. Thus, SODALITE thus develops deep learning and NLP based techniques for detecting linguistic anti-patterns and misconfiguration errors. Linguistic anti-patterns are recurring poor practices concerning inconsistencies among the naming, documentation, and implementation of an entity, have shown to be a good proxy for defect prediction [Arnaoudova 2013]. Misconfigurations or configuration errors are the violations of configuration requirements or constraints [Tianyin Xu 2013]. According to a recent report on Cloud Threat[41], 65% of cloud incidents are due to misconfigurations in provisioning and configuration scripts.

[39] https://github.com/ansible-community/ansible-lint
[40] https://github.com/rodjek/puppet-lint
[41] https://start.paloaltonetworks.com/unit-42-cloud-threat-report


© Copyright Beneficiaries of the SODALITE Project

#### 4.4.2.2 Architecture

- **Semantic Approach to TOSCA Smell Detection**

  Figure 16 illustrates some smells in TOSCA files, for instance, insecure coding practices of using admin users as the default user, and violation of a naming convention. Such smells deteriorate the quality of deployment model descriptions, and enable the exploitation of vulnerabilities in the deployed systems.

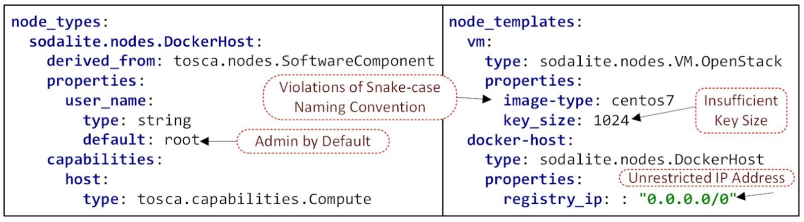


Figure 16 - Snippets of TOSCA Files Describing a Node Type and an Node Instance, Annotated with Smells

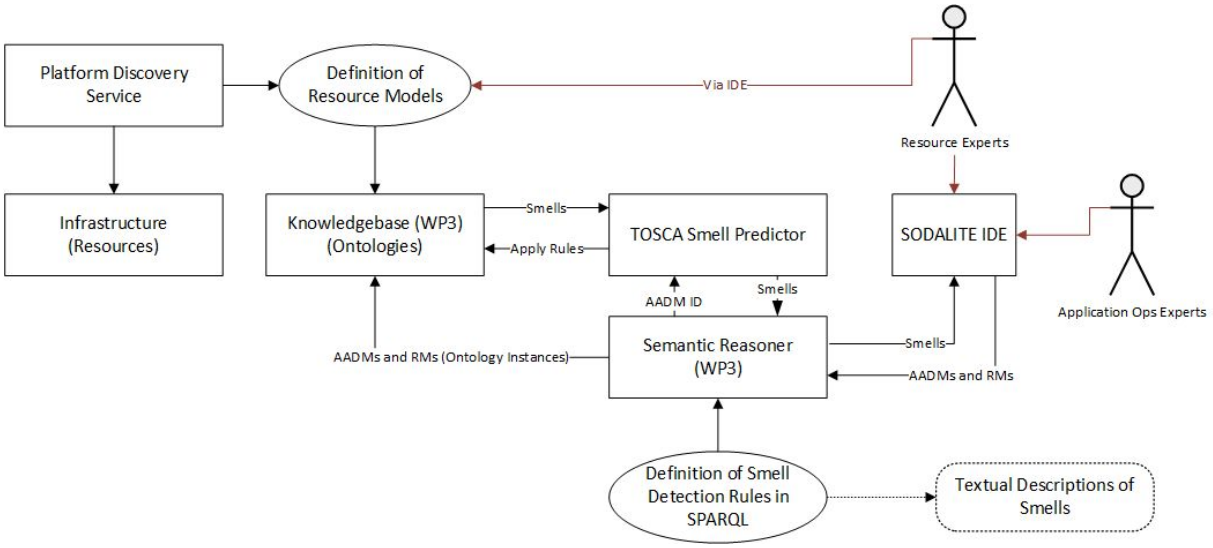


Figure 17 - An Overview of our Approach to TOSCA Smell Detection

Figure 17 shows the high-level architecture and workflow of our approach to detect the occurrences of smells in deployment model descriptions. More specifically:

- **Population of the Knowledgebase**.

  Resource Experts populate the knowledgebase by creating resource models (ontology instances representing resources/nodes in the infrastructure) using SODALITE IDE (WP3). Platform Discovery Service may (semi-)automatically update the knowledge base by creating resources models.

- **Definition of Smells Detection Rules**.

  We use the semantic rules in SPARQL to detect different smells in deployment models. There exist rules to detect common security and implementation smells. New rules can be defined to detect new types of smells.

- **Detection of Smells**.

  Application Ops Experts create the AADM instances for representing the deployment models of the applications. The AADM is automatically translated into the corresponding


© Copyright Beneficiaries of the SODALITE Project

ontological representation and is saved in the knowledgebase. The smell detection rules are applied over the ontologies in the knowledgebase to detect deployment model-level smells. If a smell is detected, the details of the smell are returned to the Application Ops Experts. The detected smells are shown in the IDE as warnings. The same flow applies to Resource Ops Experts, as they also receive warnings for their resource models.

Figure 18 shows the (abstract) rules to detect 10 TOSCA smells. The rules are implemented as SPARQL queries for specifying detection rules. Figure 19 shows an excerpt from the SPARQL query for detecting Admin by default smell. Line 4 implements the function isUser using a regex matching. Lines 5-9 retrieve the default value for a property of a node. Line 14 realizes the function *isAdmin* using the *IN* operator. The SPARQL queries for the other smells are available online in the SODALITE github repository.

| Smell | Smell Description | Abstract Detection Rule |
|---|---|---|
| Admin by default | Default users are administrative users. | isUser (x.name) $\wedge isAdmin(x.name)$ |
| Empty password | A password as a zero-length string. | isPassword(x.name) $\wedge(isEmpty(x.value) \vee isEmpty(x.defaultValue))$ |
| Hard-coded secret | Secrets such as usernames and passwords are hardcoded. | (isPassword(x.name) $\vee isUser(x.name) \vee isSecKey(x.name)) \wedge$ ((~isEmpty(x.value) $\wedge$~isVariable(x.value)) $\vee$~isEmpty(x.defaultValue)) |
| Suspicious comment | A comment includes the information indicating secrets and buggy implementations. | hasComment(x) $\wedge isSuspicious(x.comment)$ |
| Unrestricted IP address | Using "0.0.0.0" or "::" as binding IP addresses of servers | isIP(x.name) $\wedge(isInvalidBind(x.value) \vee isInvalidBind(x.defaultValue))$ |
| Insecure communication | Using insecure communication protocols, instead of their secure counterparts | (isURL(x.value) $\wedge isInsecure(x.value)) \vee$ (isURL(x.defaultValue) $\wedge isInsecure(x.defaultValue)$) |
| Weak crypto. algo. | Use of weak cryptography algorithms such as MD5 and SHA1 | hasWeakAlgo(x.value) $\vee hasWeakAlgo(x.defaultValue)$ |
| Insufficient key Size | The size of a key used by an encryption algorithm is less than the recommended key size, e.g., 2048 bits for RSA. | isCryptoKeySize(x.name) $\wedge(hasInsufficientKeySize(x.value) \vee hasInsufficientKeySize(x.defaultValue))$ |
| Inconsistent naming convention | The conventions used for naming nodes, properties, attributes, etc., are inconsistent. | (case=='CamelCase' $\rightarrow isCamelCase(x.name)) \vee (case ==' SnakeCase' \rightarrow isSnakeCase(x.name)) \vee (case ==' DashCase' \rightarrow isDashCase(x.name))$ |
| Invalid port ranges | TCP port values are not within the range from 0 to 65535. | isPort(x.name) $\wedge(outOfRange(x.value) \vee outOfRange(x.defaultValue))$ |

Figure 18 - Smells, their Descriptions, and the Abstract Detection Rules.

```
1  select distinct ?property ?propertyDef
2  where {
3      ?property DUL:classifies ?propertyDef.
4      FILTER(regex(str(?propertyDef),"user(.+?)|(.+?)?user","i")).
5      optional { # node type definitions - tier1
6            ?property DUL:hasParameter ?p .
7            ?p DUL:classifies tosca:default .
8            ?p tosca:hasDataValue ?value.
9          }.
10     optional { # node template definitions - tier0
11          ?property tosca:hasDataValue ?value.
12         }.
13     FILTER (bound(?value)).
14     FILTER (str(?value) IN ('admin', 'root'))
15 }
```

Figure 19 - Part of AdminByDefault SPARQL Query.

- **Deep Learning and NLP Approach to Linguistic Anti-Pattern Detection in IaC**
  Boosted by the emerging trend of deep learning and word embeddings for software code analysis and defect prediction, we propose a novel approach to detect linguistic anti-patterns in IaC, focusing on name-body inconsistencies in IaC code units. Figure 20 illustrates the workflow of our approach as a set of steps, which can be categorized into the following phases:


© Copyright Beneficiaries of the SODALITE Project

- **Corpus Tokenization**. Given a corpus of Ansible tasks, this phase generates token streams for both task names and bodies. To tokenize a task's body while considering its semantic properties, we build and use its abstract syntax tree (AST).
- **Data Sets Generation**. Since it is challenging to find a sufficient number of real buggy task examples that suffer from inconsistencies, we apply simple code transformations to generate buggy examples from likely correct examples. We perform such transformations on the tokenized data set and assume that most corpus tasks do not have inconsistencies. Indeed, several previous studies [Pradel 2018] in software defect prediction have successfully applied similar techniques to generate training and test data.
- **From Datasets to Vectors**. We employ *Word2Vec* [Church 2017] to convert the token sequences into distributed vector representations (code embeddings). We train a deep learning model for each Ansible module type as our experiments showed a single model does not perform well, potentially due to low token granularity. Thus, the tokenized data set is divided into subsets per module, and the code embeddings for each subset are separately generated.
- **Model Training**. This phase feeds the code embeddings to a Convolutional Neural Network (CNN) model and trains the model to distinguish between the tasks having name-body inconsistencies from correct tasks. The trained model is stored in the model repository.
- **Inconsistency Identification**. The trained models (classifiers) from the model repository are employed to predict whether the name and body of a previously unseen Ansible task are consistent or not. Each task is transformed into its corresponding vector representations, which can be consumed by a classifier.

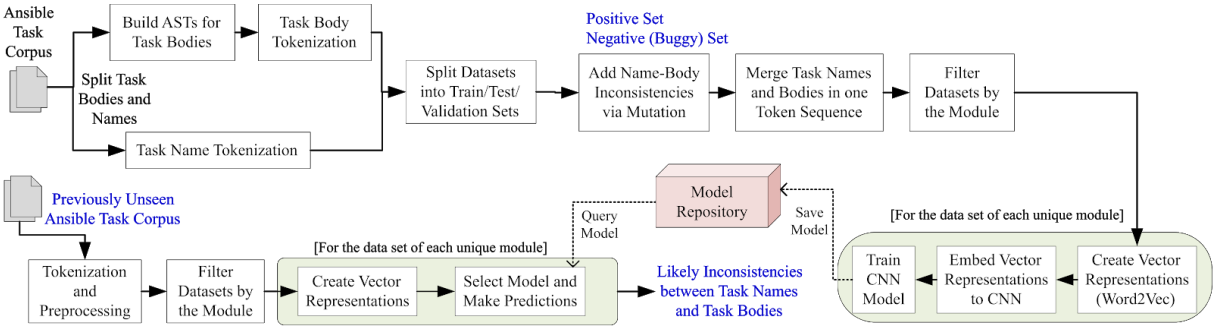


Figure 20 - Overview of our approach to detecting linguistic anti-patterns in IaC

We evaluated our approach with an Ansible dataset systematically collected from open source repositories. Figure 21 presents the inconsistency detection results for the top 10 Ansible modules in our data set. Overall, our approach yielded an accuracy ranging from 0.785 to 0.915, AUC metric from 0.779 to 0.914, and MCC metric from 0.570 to 0.830. Our approach achieved the highest performance for detecting inconsistency in the file module, where the accuracy was 0.915, the F1 score for the inconsistent class was 0.92, and the F1 score for the consistent class was 0.91. We also observed that the ROC curve, the model loss, and the accuracy plots confirm the model's good performance.


© Copyright Beneficiaries of the SODALITE Project

| Evaluation Metric/Module | | shell | command | set_fact | template | file | gather_facts | copy | service | debug | fail |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Inconsistent | Precision | 0.880 | 0.790 | 0.770 | 0.820 | 0.900 | 0.900 | 0.860 | 0.870 | 0.870 | 0.820 |
| | Recall | 0.810 | 0.840 | 0.900 | 0.940 | 0.940 | 0.830 | 0.810 | 0.760 | 0.770 | 0.690 |
| | F1 score | 0.843 | 0.814 | 0.830 | 0.876 | 0.920 | 0.864 | 0.834 | 0.811 | 0.817 | 0.749 |
| Consistent | Precision | 0.810 | 0.820 | 0.890 | 0.930 | 0.930 | 0.905 | 0.82 | 0.800 | 0.750 | 0.760 |
| | Recall | 0.890 | 0.770 | 0.750 | 0.800 | 0.890 | 0.770 | 0.870 | 0.900 | 0.860 | 0.870 |
| | F1 score | 0.848 | 0.794 | 0.814 | 0.860 | 0.910 | 0.870 | 0.844 | 0.847 | 0.801 | 0.811 |
| Accuracy | | 0.847 | 0.805 | 0.819 | 0.868 | 0.915 | 0.817 | 0.838 | 0.833 | 0.809 | 0.785 |
| MCC | | 0.697 | 0.610 | 0.649 | 0.744 | 0.830 | 0.685 | 0.678 | 0.669 | 0.625 | 0.570 |
| AUC | | 0.848 | 0.804 | 0.822 | 0.868 | 0.914 | 0.848 | 0.838 | 0.830 | 0.814 | 0.779 |

Figure 21 - Classification results for the top 10 used Ansible modules.

#### 4.4.2.3 Features

The IaC defect prediction tools collectively provides the capabilities of:

- detecting security and implementation smells in TOSCA and Ansible scripts
- suggesting fixes for detected TOSCA smells
- detecting linguistic anti-patterns in Ansible/IaC scripts
- detecting misconfiguration errors in Ansible /IaC scripts

#### 4.4.2.4 Status

- **Overall Architecture and Methodology of Analytics and Semantic Support**
  - We published the design and workflow of our Analytics and Semantic Support in ESOCC 2020 [Kumara1 2020].
- **Semantic Approach to TOSCA Smell Detection**
  - We developed the support for key common types of security and implementation smells reported in the IaC literature. The defection prediction tool was integrated with SODALITE IDE.
  - We published our semantic approach to defect prediction at the 10th International Conference on Web Intelligence, Mining and Semantics [Kumara2 2020], and are currently extending the conference paper to a journal paper.
- **Deep Learning and NLP Approach to Linguistic Anti-Pattern Detection in IaC**
  - We developed the support for detecting the inconsistencies between task names and task bodies in Ansible playbooks/roles.
  - We published our approach for linguistic anti-pattern detection at the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation [Borovits 2020]. We have also received a journal invitation for the paper at Empirical Software Engineering (EMSE) journal, and are currently extending our initial work for this journal.

#### 4.4.2.5 Next steps

- Develop an NLP and DL based approach to support detecting misconfigurations in IaC scripts (M33)
- Improving the support for recommending fixes for TOSCA smells (M30)

# 5 Extension of the existing components

## 5.1 Image builder

SODALITE uses the *Image Builder* component to prebuild application images for targeting an OS virtualizer such as Docker or Singularity. The *Image Builder* component itself is a dockerized REST API encapsulation of the *xOpera* lightweight orchestrator and a *TOSCA/Ansible* blueprint that is executed by the orchestrator and can be configured to run different image building workflows enabling the user to build the application from source or tar images and push the created image to a Docker registry. The image building workflows for building runtime images are running prior to


© Copyright Beneficiaries of the SODALITE Project

deployment of the TOSCA blueprint, before the orchestrator starts with the execution of the blueprint deployment e.g., provisioning the infrastructure and deployment of the application. The encapsulation of the xOpera lightweight orchestrator and TOSCA blueprints into the REST API enables the image building functionality to be accessible from any component in SODALITE framework or be just reused in a separate blueprint if needed. The extendable nature of these TOSCA blueprints provides a high level of reusability of the code for supporting the image building process. Image builder also supports session handling and authentication/authorization by JWT tokens making it easy to integrate with Identity and Access Management providers.

### 5.1.1 Improvements

According to our plans, significant improvements were added since the initial image-builder release at the end of the first year of the project. In this section we focus on most prominent ones with regards to features, implementation and software quality.

During the second year of the project tests for the developed features were significantly improved adding quality and ensuring to not break the already developed features. In this year, the TOSCA standard version was upgraded from 1.2 to 1.3 so the blueprints were also updated to be inline with the latest TOSCA developments. An important innovative feature was implemented: giving the user the possibility to create multiple image variants in a single image building workflow run.

Additionally, CI/CD process was aligned with the SODALITE CI/CD streamlined and integrated development process described in more details in the deliverable D6.3, enabling a simple and informative management of the testing and deployment to the testbed, pushing images to docker hub repository guided through simple semantic version tagging and git branch naming compliance.

Image builder exposes its functionalities through both multifunctional CLI and REST API, and can be also used in a CI/CD scenario.

Finally the NGINX[42] image with saved configuration for image-builder dockerized API was replaced by the simple configuration of Traefik[43] proxy enabling a more efficient possible transition to k8s deployment.

The code and extensive information on how to build and use the Image builder is provided in the github repository: https://github.com/SODALITE-EU/image-builder

### 5.1.2 Code Quality

Since the first image-builder release automatic code quality checks were introduced using the online SonarCloud tool. Several corrections of the code were applied after enabling the SonarCloud code analysis of the image builder component. One of the most important being the extension of the unit tests for code coverage and reduction of code repetitions.

At the time of this writing the quality of the code in the image-builder github repository was solid with a code coverage of 73.6%, no bugs and vulnerabilities and a small percentage of code duplications as shown in Figure 22. In the next releases we will focus on reducing the code smells and on reviewing the security hotspots (currently SonarCloud highlights 5 security hotspots):

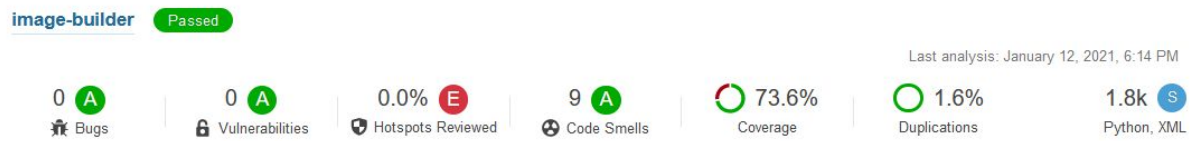


Figure 22 - Current code quality values for the Image Builder.

[42] https://www.nginx.com/

[43] https://traefik.io/ Cloud-Native Networking Stack That Just Works.


© Copyright Beneficiaries of the SODALITE Project

### 5.1.3 Next steps

Improving security and integration with IDE is one of the most prominent next steps in the development of the image-builder component. Integration with IDE would enable an AOE to create the container image for a component directly from the IDE instead of using the image builder REST API or CLI interface.

Additionally the support of building multi-architecture images remains a top priority feature for this component. This feature would enable the user to inject code changes and different configurations for specific build target platforms.

## 5.2 IaC-Blueprint-Builder

The IaC Blueprint Builder internally generates the TOSCA blueprint by transforming the AADM JSON, passed as input by the IDE in the form of REST API which is passed to the Abstract Model Parser and eventually the Application Optimiser optimises the application for a given target platform based on the optimisation options selected.

### 5.2.1 Improvements

The initial version of Iac-Blueprint-Builder was working and providing the TOSCA blueprints as required but there were few issues related to the output produced like broken blueprints at some parts or missing required fields. Moreover, the code was not properly organized and structured which might cause trouble in future maintenance.

During the second year of the project, the code was refactored so that it can be more efficient to run and maintain, and the documentation improved. The initial issues concerning the generated output were resolved and few missing functionalities were added to take into account the evolution of the AADM input received by the component. The integration with MODAK for the optimization purpose has been implemented.

The code is provided in the repository: https://github.com/SODALITE-EU/iac-blueprint-builder

### 5.2.2 Code Quality

The Iac-blueprint-builder has been implemented in Python. In the last year several corrections have been made in the code as part of refactoring the parsing file in specific.

Some bugs and vulnerabilities were also fixed from the earlier version. The test coverage has improved from 54% to 87.5%. The code smells and duplications remain the same, code duplication being 0% as shown in Figure 23.

In the next releases we will focus on reducing the code smells and on reviewing the security hotspots (currently SonarCloud highlights 2 security hotspots).

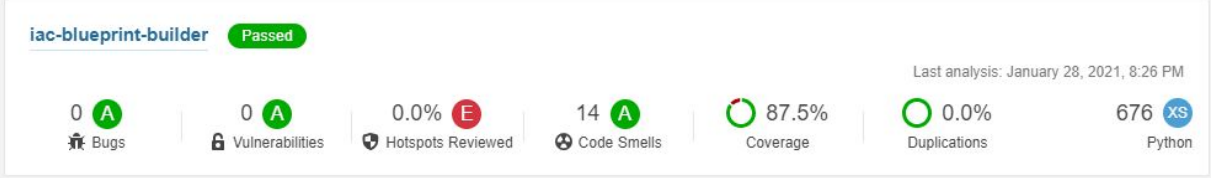


Figure 23 - Current code quality values for the Iac Blueprint Builder.

### 5.2.3 Next steps

Adding authentication/authorization functions to improve the security of the repository should be added.


© Copyright Beneficiaries of the SODALITE Project

Additionally, proper unit tests should be framed so that it is easier to check in future that all the functionalities are working well.

## 5.3 Prediction service

*Bug Predictor and Fixer* and *Predictive Model Builder* are the two components of the prediction service. They share the same github repositories.

Bug Predictor and Fixer detects the smells and bugs in TOSCA and Ansible artifacts and suggests corrections or fixes for some of the smells/bugs. For smells/bugs detection, Bug Predictor and Fixer uses a model built by the Predictive Model Builder. This last one uses a rule-based model for detecting implementation and security smells in Ansible. The semantic reasoning over the SODALITE ontologies, developed in WP3, is performed.

### 5.3.1 Improvements

At M12, Bug Predictor and Fixer supported the rule-based approach to detecting smells in Ansible. The rules are Ansible Lint[44] rules. In the second year, we added new rules to detect more smell types. At M12, Bug Predictor and Fixer also had the initial implementation of the semantic approach to detecting TOSCA smells. In the second year, we significantly improved the initial support with the capabilities to detect more smells and the integration with SODALITE IDE. The improved smell detection was also published at a scientific conference. Section 4.4.2 describes this improved support under the feature *Semantic Approach to TOSCA Smell Detection*.

Since M12, the Predictive Model Builder has been extended with the NLP and deep learning-based models for detecting linguistic smells in Ansible (see Section 4.4.2).

### 5.3.2 Code Quality

The Bug Predictor and Fixer has been implemented in Java and Python, and consequently, have two separate github repositories. The implementations of the new features mentioned in Section 4.4.2 are also part of Bug Predictor and Fixer and Predictive Model Builder. As shown in Figure 24 and 25, the Python part of Bug Predictor and Fixer has 34.9% code coverage, 57 code smells and 3.0% code duplications, and the Java part has 52% code coverage, 33 code smells and 2.5% code duplications. In the next releases we will focus on reducing the code smells, removing duplicate code by refactoring the code base, and improving code coverage by adding more unit tests.

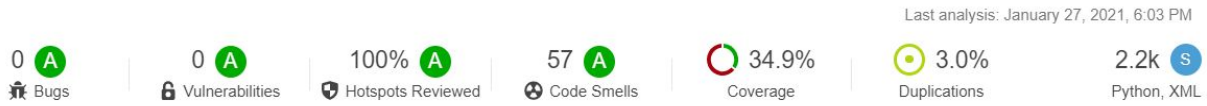


Figure 24 - Current code quality values for the Bug Predictor and Fixer - Python part.


Figure 25 - Current code quality values for the Bug Predictor and Fixer - Java part.

[44] https://github.com/ansible-community/ansible-lint


© Copyright Beneficiaries of the SODALITE Project

### 5.3.3 Next steps

- Integration of Ansible smell detection with SODALITE IDE

## 5.4 IaC Quality Assessor

This component can calculate different software quality metrics for IaC artifacts. We develop this component in collaboration with the RADON project (see [Dalla Palma 2020]).

### 5.4.1 Improvements

Compared with M12, during the second year, new metrics were added, and the component CI/CD was integrated with Sonarcloud. A REST API was developed.

### 5.4.2 Code Quality

The IaC Quality Assessor has 51.8% code coverage, and 19 code smells. . In the next releases we will focus on reducing the code smells and improving code coverage by adding more unit tests.


Figure 26 - Current code quality values for the IaC Quality Assessor.

### 5.4.3 Next steps

- Add workflow level quality metrics for Ansible playbooks and roles

## 5.5 Topology Verifier

This component verifies the constraints over the structures of the TOSCA blueprints. Topology Verifier uses the verification capabilities provided by Openstack Tosca Parser[45] to detect the violations of the syntax of blueprints with respect to the TOSCA specification. However, WP3 (Semantic Reasoner) implements the support for verifying the requirements of the nodes, the node-relationships, the capabilities of the nodes, and node substitutability.

### 5.5.1 Improvements

During the second year, some bug fixes were done and the component CI/CD was also integrated with Sonarcloud.

### 5.5.2 Code Quality

The Topology Assessor has 25.4% code coverage, and 0 code smells. In the next releases we will focus on improving code coverage by adding more unit tests.


Figure 27 - Current code quality values for the Topology Verifier.

### 5.5.3 Next steps

- Improve and use XOpera Parser, which supports TOSCA 1.3 version

[45] https://github.com/openstack/tosca-parser


© Copyright Beneficiaries of the SODALITE Project

## 5.6 Provisioning Workflow Verifier

This component verifies the control flow constraints over the provisioning workflow of the application using Petri Net, one of the widely used techniques for verifying workflows. The workflow is described in the Ansible scripts in terms of tasks, roles, plays, and variables. The implementation uses the Petri Net support provided by *PM4Py*[46] process mining library.

### 5.6.1 Improvements

During the second year, some bug fixes were done, and the component CI/CD was also integrated with Sonarcloud.

### 5.6.2 Code Quality

The Topology Assessor has 22.0% code coverage, and 0 code smells. In the next releases we will improve the code coverage by adding more unit tests.


Figure 28 - Current code quality values for the Provisioning Workflow Verifier.

### 5.6.3 Next steps

- Improve translation from Ansible playbooks to Petri-Net

## 5.7 IaC Verifier

This component provides an unified REST API (Facade) for the verification capabilities of Topology Verifier and Provisioning Workflow Verifier.

### 5.7.1 Improvements

During the second year, the initial Java based implementation was replaced by a Python-based implementation. Moreover, some bug fixes were done, and the component was also integrated with Sonarcloud.

### 5.7.2 Code Quality

The IaC Verifier has a Python REST API and does not have unit tests. In the next releases we will focus on improving code coverage by adding unit tests.


Figure 29 - Current code quality values for IaC Verifier

### 5.7.3 Next steps

- Update the API if the capabilities of Topology Verifier and Provisioning Workflow Verifier are updated.
- Improved integration with SODALITE IDE

[46] https://pm4py.fit.fraunhofer.de/


© Copyright Beneficiaries of the SODALITE Project

# 6 Updated IaC Management Layer Development Plan

Tables 1 and 2 provide a summary of the features that will be developed as part of the IaC Management Layer by M30 and M36 respectively. These features have been presented also in the previous sections of this deliverable.

Besides feature development, in the last part of the project careful attention will be posed to increasing the quality of the release artifacts in terms of code quality metrics and also in terms of their usability and flexibility, based on the feedback gathered by case study owners and other users.

| Component | Planned features |
|---|---|
| Abstract Model Parser and IaC Blueprint Builder | Adding authentication/authorization functions to improve the security of the repository should be added.<br>Adding proper unit tests so that it is easier to check in future that all the functionalities are working well. |
| Runtime Image Builder and Concrete Image Builder | Integration with IDE,<br>add multi-architecture build variants |
| Image Registry | N/A |
| Application Optimiser | Cloud-system support |
| IaC Taxonomies | IaC Smell and Bug Taxonomy (Final Versions) |
| IaC Bug Prediction and Correction,<br>Verification and Quality Assurance | Suggesting Fixes for TOSCA Smells (Basic Support),<br>Misconfiguration Detection for Ansible,<br>Control Flow Ansible Metrics,<br>Control Flow Verification of Ansible (Improved Support) |
| Platform Discovery Service | Initial integration with semantic reasoner,<br>Improved integration with IDE,<br>Initial kubernetes discovery |
| Ansible support | To complete the support to the creation of Ansible playbook, we intend to integrate as part of the IDE also the possibility to search through the Ansible Modules available online and to select the right one to be used within an abstract playbook. |

Table 2 - IaC Management Layer Release Plan for M30.

| Component | Planned features |
|---|---|
| Abstract Model Parser and IaC Blueprint Builder | Bug fixing |
| Runtime Image Builder and Concrete Image Builder | Improved workflows for the image building process |


© Copyright Beneficiaries of the SODALITE Project

| | |
|---|---|
| Image Registry | N/A |
| Application Optimiser | Big-data analytics application support |
| IaC Taxonomies | Validated Final Taxonomies |
| IaC Bug Prediction and Correction, Verification and Quality Assurance | Suggesting for Fixes for TOSCA Smells (Improved Support), Improved Integration of Bug/Smell Detection and Verification with IDE |
| Platform Discovery Service | Support for TOSCA changes, Improved versions of TOSCA for Openstack, AWS and kubernetes discovery |
| Ansible support | If possible, given the project resource constraints and other commitments, develop a backward transformation from Ansible playbooks to the corresponding abstract ones. This feature could be useful to simplify the modification of preexisting playbooks within the context of the SODALITE framework. |

Table 3 - IaC Management Layer Release Plan for M36.

# 7 Conclusion

This deliverable has reported on the intermediate release (M24) of the SODALITE IaC Management Layer, emphasizing the main new features that have been incorporated since the initial release reported in D4.1, but also reporting on the progress achieved on existing components.

The quality of the developed components and their interoperability have been largely improved and will be subject to further improvement in the last project year.

The main achievements reported for this year concern: i) the addition of the platform discovery feature that significantly reduces the work of Resource Experts thanks to its ability to automatically discover new resources and encode them as Resource Models; ii) the support to the creation of Ansible scripts, which can be optionally used by Resource Experts is case they need to define new Ansible Playbooks; iii) Modak, the software-defined optimisation framework for containerised HPC and AI applications; and iv) the improved support for detecting code smells and prevent bugs.

The main challenges foreseen for the last year of the project concern the consolidation of all components and improving their integration with the modeling environment. Moreover, the software-defined optimisation framework will be extended to cope also with cloud-based systems.


© Copyright Beneficiaries of the SODALITE Project

## References

[Rutkowski20] Matt Rutkowski, Chris Lauwers, Claude Noshpitz, Calin Curescu, TOSCA Simple Profile in YAML Version 1.3, OASIS Standard. February 2020. https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.pdf

[D4.1] Dragan Radolović (XLAB) Nejc Bat (XLAB) Elisabetta Di Nitto (POLIMI) Mehrnoosh Askarpour (POLIMI) Karthee Sivalingam (CRAY) Indika Kumara (JADS/UVT) Panagiotis Mhtzias, Georgios Meditskos (CERTH) Kalman Meth (IBM). D4.1 IaC Management initial version. SODALITE Technical report 2020. https://sodalite.eu/reports/d41-iac-management-initial-version.

[D2.2] Luciano Baresi (POLIMI), Elisabetta Di Nitto (POLIMI), Dragan Radolović (XLAB), Alexander Maslennikov (XLAB), Kalman Meth (IBM), Yosu Gorroñogoitia (ATOS), Kamil Tokmakov (USTUTT), Indika Kumara (JADS), Giovanni Quattrocchi (POLIMI), Zoe Vasileiou, Anastasios Karakostas, Savvas Tzanakis (CERTH). D2.2 Requirements, KPIs, evaluation plan and architecture - Intermediate version. SODALITE Technical report 2021, to be published.

[D2.4] Luciano Baresi (POLIMI), Elisabetta Di Nitto (POLIMI), Kamil Tokmakov (USTUTT), Anastasios Karakostas (CERTH), Stefanos Vrochidis (CERTH), Dragan Radolović (XLAB). D2.4 Guidelines for contributors to the SODALITE framework. SODALITE Technical report 2020. https://sodalite.eu/reports/d24-guidelines-contributors-sodalite-framework.

[D5.2] Jesús Gorroñogoitia (Atos), Jorge Fernández Fabeiro (Atos), Lucas Pelegrin Caparrós (Atos), Indika Kumara (JADS/UVT), Dragan Radolović (XLAB), Nejc Bat (XLAB), Kamil Tokmakov (USTUTT), Kalman Meth (IBM), Giovanni Quattrocchi (POLIMI). D5.2 - Application deployment and dynamic runtime - intermediate version. SODALITE Technical report 2021, to be published.

[D6.3] Kamil Tokmakov (USTUTT), Ralf Schneider (USTUTT), Dennis Hoppe (USTUTT), Kalman Meth (IBM), Elisabetta Di Nitto (POLIMI), Paul Mundt (ADPT), Airán González Gómez (ATOS), Yosu Gorroñogoitia (ATOS), Dragan Radolović (XLAB), Mihael Trajbarič (XLAB), Piero Fraternali (POLIMI), Rocio Nahime Torres (POLIMI), Vasileios-Rafahl Xefteris, Savvas Tzanakis(CERTH), Karthee Sivalingam (CRAY), Indika Kumara (UVT/JADS), Lucas Pelegrin (ATOS), Saloni Kyal (POLIMI), Giovanni Quattrocchi (POLIMI). D6.3 Intermediate implementation and evaluation of the SODALITE platform and use cases. SODALITE Technical report 2021, to be published.

[D6.6] Kalman Meth (IBM), Indika Kumara (JADS), Zoe Vasileiou, Vasileios-Rafahl Xefteris, Savvas Tzanakis, Anastasios Karakostas, Spyridon Symeonidis (CERTH), Yosu Gorroñogoitia, Lucas Pelegrin (ATOS), Giovanni Quattrocchi (POLIMI), Dragan Radolović, Nejc Bat (XLAB), Alfio Lazzaro (HPE), Saloni Kyal (POLIMI) D6.6 SODALITE Framework - Second Version. SODALITE Technical report 2021, to be published.

[Schliephake 2012] Michael Schliephake and Erwin Laure. 2012. Towards Improving the Communication Performance of CRESTA's Co-Design Application NEK5000. In 2012 SC Companion: High Performance Computing, Networking Storage and Analysis. IEEE, 669–674.

[SonarCloud1] Java static code analysis - Technical manual, https://rules.sonarsource.com/java/type/Bug

[SonarCloud2] Nicolas Harraudeau, How SonarCloud finds bugs in high-quality Python projects, blog article https://blog.sonarsource.com/sonarcloud-finds-bugs-in-high-quality-python-projects

[Sharama 2018] Sharma, Tushar, and Diomidis Spinellis. "A survey on software smells." Journal of Systems and Software 138 (2018): 158-173

[Wettinger 2016] J Wettinger, U Breitenbücher, O Kopp, F Leymann. Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel. Future Generation Computer Systems 56, 317-332. 2016


*© Copyright Beneficiaries of the SODALITE Project*

[Tamburri 2019] Tamburri, D.A., Van den Heuvel, WJ., Lauwers, C. et al. TOSCA-based Intent modelling: goal-modelling for infrastructure-as-code. SICS Software-Intensive Cyber-Physical Systems. 34, 163–172 (2019).

[Noudohouenou 2014] J. Noudohouenou and W. Jalby, "Using static analysis data for performance modeling and prediction," 2014 International Conference on High Performance Computing & Simulation (HPCS), Bologna, 2014, pp. 933-942, doi: 10.1109/HPCSim.2014.6903789.

[Brogi 2016] A. Brogi, P. Cifariello, J. Soldani: DrACO: Discovering Available Cloud Offerings. Computer Science: Research and Development, Springer, 2016 https://link.springer.com/article/10.1007%2Fs00450-016-0332-5

[Bhattacharjee 2018] A. Bhattacharjee, Y. Barve, A. Gokhale and T. Kuroda, "A Model-Driven Approach to Automate the Deployment and Management of Cloud Services," 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, 2018, pp. 109-114, doi: 10.1109/UCC-Companion.2018.00043.

[Bhattacharjee 2017] Bhattacharjee, A., Y. Barve, T. Kuroda, and A. Gokhale. CloudCAMP: A Model-driven Generative Approach for Automating Cloud Application Deployment and Management. Report by Institute for Software Integrated Systems, Vanderbilt University. ISIS-17-105. 2017.

[Barve 2018] Y. Barve et al. UPSARA: A Model-Driven Approach for Performance Analysis of Cloud-Hosted Applications. In: 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC). 2018, pp. 1{10. doi: 10.1109/UCC.2018.00009.

[Rutkowski 2020] Matt Rutkowski, Chris Lauwers, Claude Noshpitz, Calin Curescu. TOSCA Simple Profile in YAML Version 1.3. OASIS Standard. February 2020. https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.pdf

[Chiba 2019] Tatsuhiro Chiba, Rina Nakazawa, Hiroshi Horii, Sahil Suneja, and SeetharamiSeelam. 2019. ConfAdvisor: A Performance-centric Configuration Tuning Framework for Containers on Kubernetes. In 2019 IEEE International Conference onCloud Engineering (IC2E). IEEE, 168–178.

[Kubernetes 2021] Kubernetes, an open-source system for automating deployment, scaling, and management of containerized applications. https://kubernetes.io/, 2021

[AWS 2020] AWS Compute Optimizer. 2020. https://aws.amazon.com/compute-optimizer/

[de Bayser 2017] Maximilien de Bayser and Renato Cerqueira. 2017. Integrating MPI with Dockerfor HPC. In2017 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 259–265.

[McMillan 2018] Scott McMillan. 2018. Making Containers easier with HPC container maker. In Proceedings of the SIGHPC System Professionals Workshop 2018, Dallas, TX, USA.

[Höb 2020] Maximilian Höb and Dieter Kranzlmüller. 2020. Enabling EASEY deployment of containerized applications for future HPC systems. arXiv:2004.13373 [cs.DC]

[D3.3] Alfio Lazzaro; Karthee Sivalingam; Nina Mujkanovic; Indika Kumara; Piero Fraternali; Rocio Nahime Torres; Giovanni Quattrocchi; Kamil Tokmakov; Ralf Schneider; Paul Mundt. D3.3 Prototype of application and infrastructure performance models. SODALITE technical report 2020. https://sodalite.eu/reports/d33-prototype-application-and-infrastructure-performance-models

[Arnaoudova 2013] Arnaoudova, Venera, et al. "A new family of software anti-patterns: Linguistic anti-patterns." 2013 17th European Conference on Software Maintenance and Reengineering. IEEE, 2013.

[Pradel 2018] Pradel, Michael, and Koushik Sen. "DeepBugs: A learning approach to name-based bug detection." Proceedings of the ACM on Programming Languages 2.OOPSLA (2018): 1-25.


*© Copyright Beneficiaries of the SODALITE Project*

[Church 2017] Church, Kenneth Ward. "Word2Vec." Natural Language Engineering 23.1 (2017): 155-162.

[Kumara1 2020] Kumara, Indika, et al. "Quality Assurance of Heterogeneous Applications: The SODALITE Approach." European Conference on Service-Oriented and Cloud Computing (ESOCC 2020), Volume 2. Springer, Cham, 2020 (in print).

[Kumara2 2020] Kumara, Indika, et al. "Towards Semantic Detection of Smells in Cloud Infrastructure Code." Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics. 2020.

[Borovits 2020] Borovits, Nemania, et al. "DeepIaC: deep learning-based linguistic anti-pattern detection in IaC." Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation. 2020.

[Dalla Palma 2020] Dalla Palma, Stefano, Dario Di Nucci, and Damian A. Tamburri. "AnsibleMetrics: A Python library for measuring Infrastructure-as-Code blueprints in Ansible." SoftwareX 12 (2020): 100633.

[Tianyin Xu 2013] Xu, Tianyin, et al. "Do not blame users for misconfigurations." Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. 2013.

[Rahman1 2019] Rahman, Akond, Chris Parnin, and Laurie Williams. "The seven sins: security smells in infrastructure as code scripts." 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019.

[Schwarz 2018] Schwarz, Julian, Andreas Steffens, and Horst Lichter. "Code Smells in Infrastructure as Code." 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC). IEEE, 2018.

[Sharma 2016] Sharma, Tushar, Marios Fragkoulis, and Diomidis Spinellis. "Does your configuration code smell?." 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). IEEE, 2016.

[Rahman2 2020] Rahman, Akond, et al. "Gang of eight: A defect taxonomy for infrastructure as code scripts." Proceedings of the 42nd International Conference on Software Engineering, ICSE. Vol. 20. 2020.

[Alon 2019] Alon, Uri, et al. "code2vec: Learning distributed representations of code." Proceedings of the ACM on Programming Languages 3.POPL (2019): 1-29.

[Liu 2019] Liu, Kui, et al. "Learning to spot and refactor inconsistent method names." 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019.

[Sharma 2018] Sharma, Tushar, and Diomidis Spinellis. "A survey on software smells." Journal of Systems and Software 138 (2018): 158-173.


*© Copyright Beneficiaries of the SODALITE Project*

# Appendix - Ansible Implementation Metamodel

## Notes about the notation

The “1 to 1” relations are all represented like attributes in the entities, to improve readability. The question mark “?” written before one of those attributes means that the relation is “1 to 0..1”. For example, the Base entity could be in relation with 0 or 1 Connection entity. In all the other cases, the relation is represented with the classic arrow with the indication of the cardinality of the relation.

The metamodel is split among different figures, to improve readability as well.

This metamodel is strongly coupled with the model generated by the xtext grammar implemented in the IDE, in the sense that there is almost a 1 to 1 relation between the entities of this metamodel and the entities belonging to the model generated by the xtext grammar.

Because of the strong relation between the metamodel and the grammar, and because the Ansible grammar defined in the IDE is used together with the RM grammar defined in the IDE as well, this metamodel contains references to elements belonging to the RM grammar. When this happens, those elements are referenced with a “RM_” prefix.

## The metamodel

In the following the figures reporting the metamodel are shown, with an explanation of the main aspects of it.

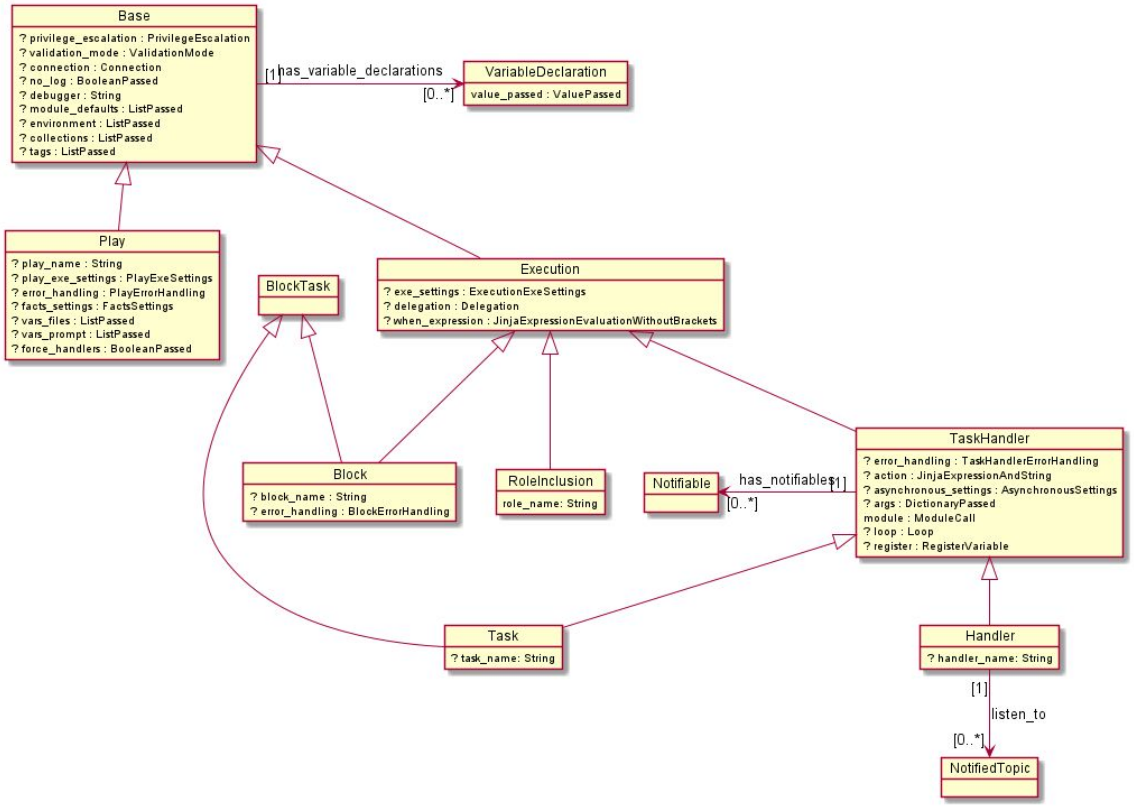


Figure 30 - Ansible main elements hierarchy.


© Copyright Beneficiaries of the SODALITE Project

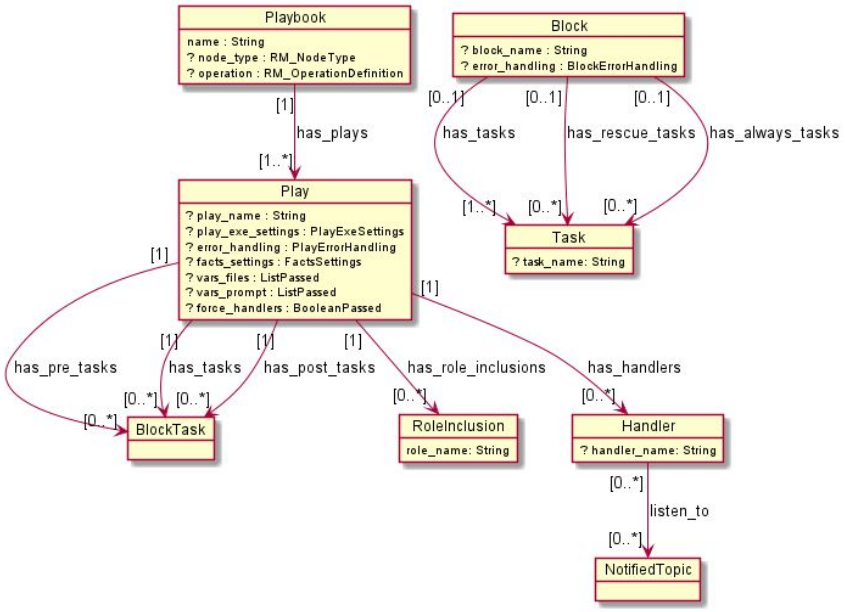


Figure 31 - Playbooks, Play and Block.

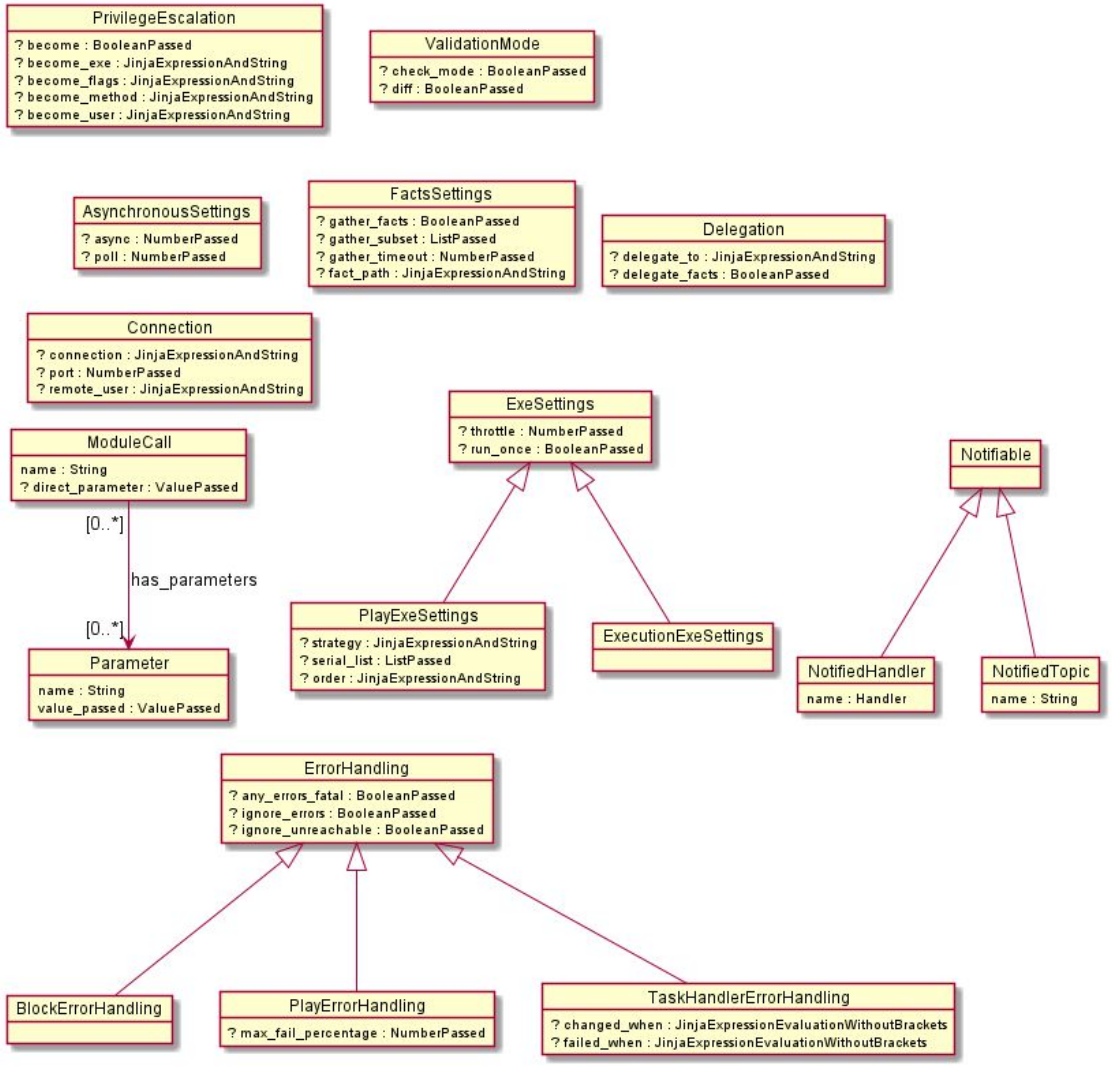


Figure 32 - Groups of parameters.


*© Copyright Beneficiaries of the SODALITE Project*

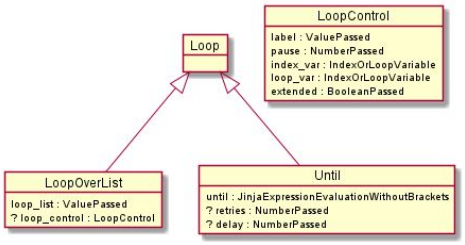


Figure 33 - Loops.

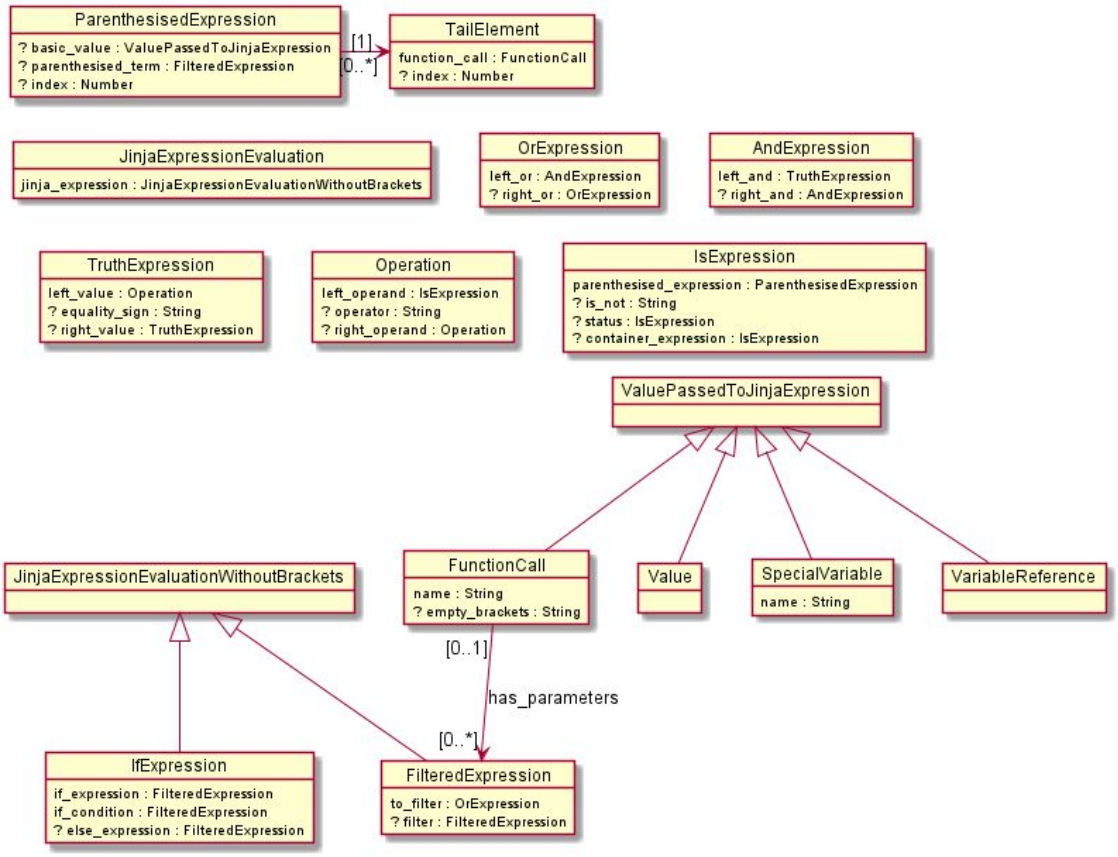


Figure 34 - Expressions.


© Copyright Beneficiaries of the SODALITE Project

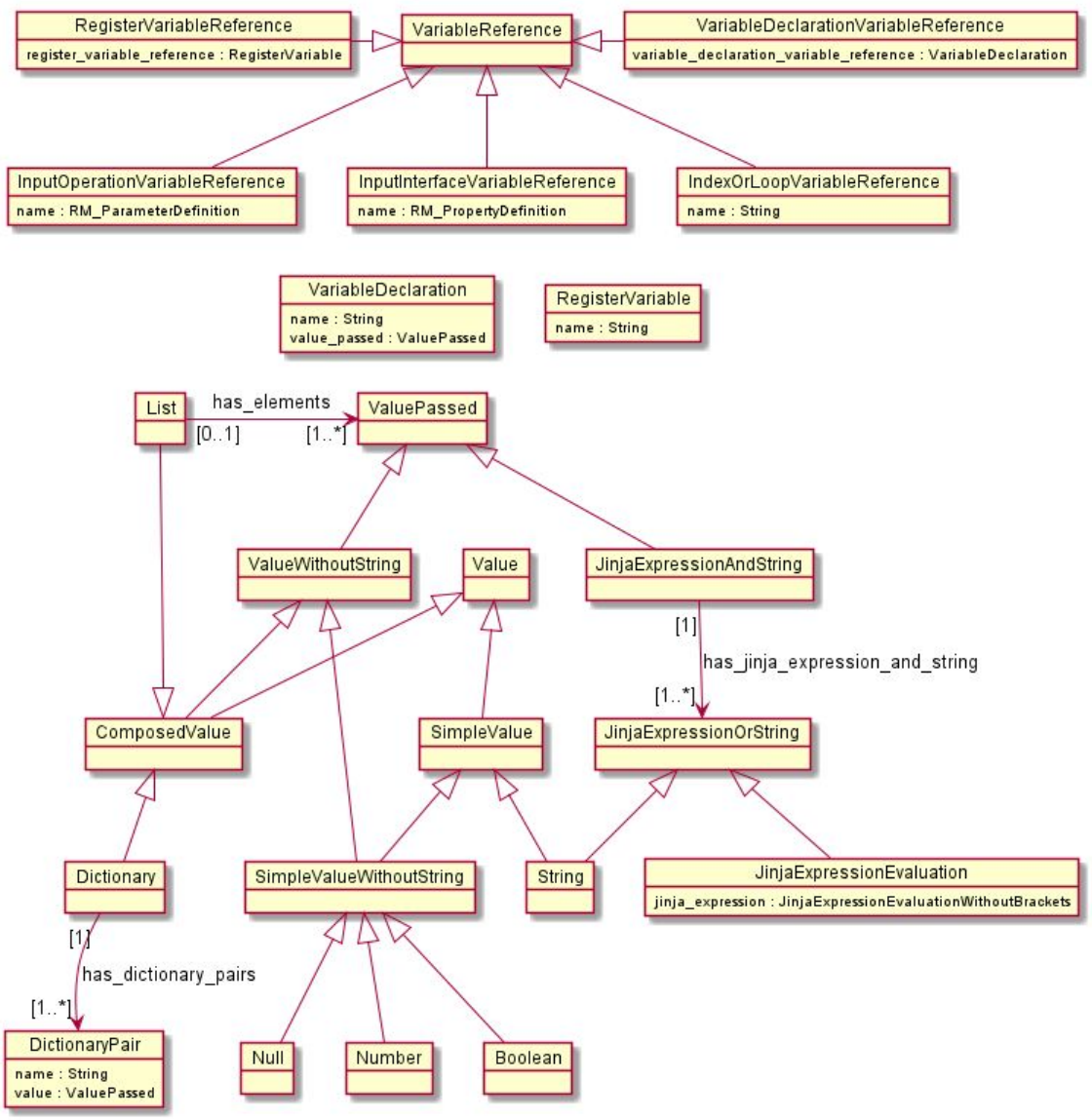


Figure 35 - Variables and values.

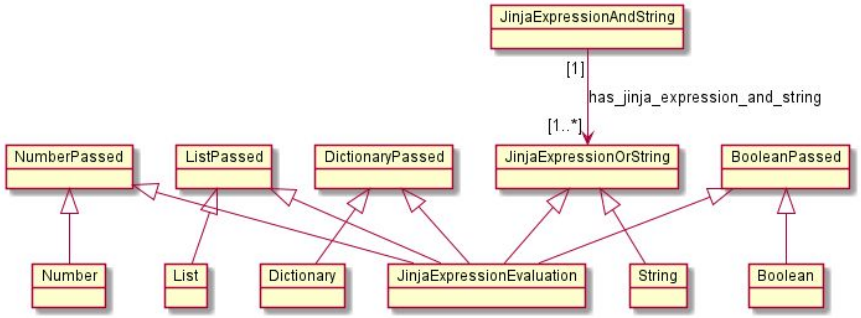


Figure 36 - Types of parameters passed by a TOSCA interface to an Ansible Playbook.


© Copyright Beneficiaries of the SODALITE Project

- *Base*: This entity was modelled by taking inspiration from the official ansible repository (https://github.com/ansible/ansible/tree/devel/lib/ansible/playbook) All the main entities of the model inherit from the Base entity, which groups the attributes that they have in common.
- *Play*: It all starts with the Playbook, which is a collection of plays (as it can be seen in the second image). The playbook is the artifact to which the interface's operation of the TOSCA Resource Model refers. The playbook can contain one or more plays, and each play contains an ordered list of operations (BlockTask) to be executed. The operations (BlockTask) could be "normal" operations, "pre_tasks" or "post_tasks". It's just a matter of order of execution: "pre_tasks" are executed before the roles included, the "normal" operations after the roles included, then the triggered handlers and then the "post_tasks" (if some concepts named here are not clear, the reader can find an explanation in the points that follow).
  Play inherits from Base, so it contains all the Base's attributes plus other attributes shown in the first image.
- *BlockTask*: The operations contained in a Play can be blocks or tasks. The BlockTask entity was modelled to capture in a unique entity all the operations that can be run in a Play, so Block and Task inherit from BlockTask.
- *Execution*: The Execution entity was modelled to capture the entities that represent some operation that can be run. The main point is grouping in this Execution entity all the attributes that they have in common (apart from the Base's attributes of course, from which Execution inherits).
- *Block*: As it can be seen in the second image, the Block is a collection of tasks (an ordered list). Apart from the "normal" tasks that it can contain, it can also have an ordered list of "always" tasks and "rescue" tasks. The "always" tasks are run in any case, even in case of errors. The "rescue" tasks are run to recover from error states.
- *RoleInclusion*: a play could contain an ordered list of roles identifiers to be executed. Executing a role means executing a set of operations, and a role can be for example downloaded from the internet (usually from Ansible Galaxy). For example, a role could be named "mongodb_server", and executing that role means making the host on top of which it runs a mongodb server. So, the "mongodb_server" role contains all the operations that, if executed, make the host a mongodb server. RoleInclusion however doesn't model the role itself, but its inclusion in the play. So it contains the identifier of the role (the name attribute) and other attributes inherited from Execution that can be used to set how the inclusion is done.
- *TaskHandler*: Since Task and Handler entities share a lot of attributes, the TaskHandler entity was modelled in order to group all of them.
- *Task*: It is the basic operation that can be run in a Play. The main thing to consider is that it executes a module (the same happens with the Handler, so the ModuleCall is contained in the TaskHandler entity). A module is a predefined operation that can be run and needs some inputs to be executed. The task passes those inputs to the module and executes it (to make an analogy: it's like if the module was an implemented function, and the task calls this function passing it some parameters). A list of the available modules can be found here: https://docs.ansible.com/ansible/2.9/modules/list_of_all_modules.html
  The inputs passed to a module are modelled as parameters (with the Parameter entity), that have an identifier and a value passed. However, it can happen that the input to the module is given without an explicit parameter with an identifier. It happens for example with the "shell" module[47], in which the shell command is passed without an explicit named

[47] https://docs.ansible.com/ansible/2.9/modules/shell_module.html#shell-module


© Copyright Beneficiaries of the SODALITE Project

parameter. The direct_parameter attribute modelled in this metamodel has the purpose of being used in situations like this, instead of the Parameter entity.

- *Handler*: Very similar to the Task, but it's executed when an event occurs. The TaskHandler entity can be associated with some Notifiable elements, as it can be seen in the first image. A Notifiable element, as it can be seen in the third image, could be a Handler or a topic. So TaskHandler (tasks and handlers) can notify one or more handlers (or topics, as it will be explained shortly). It means that when the task/handler finishes its execution, the notified handlers are executed as consequence.
  A Handler can also listen to a specific topic. If the Notifiable element associated with the task/handler is a topic, then all the handlers listening to that topic are executed consequently.
- *Loop*: tasks and handlers (so the TaskHandler entity) have the "loop" attribute. A Loop makes the execution of the task/handler iterate. As it can be seen in the fourth image, a Loop can be a LoopOverList entity or an Until entity. In the case of the LoopOverList entity, the task/handler is re-executed once for each element contained in the input given by the LoopOverList. The LoopOverList entity may also contain a LoopControl, with additional settings for the execution of the loop. If the Loop is an Until entity instead, the task/handler is re-executed until a certain condition is met.
- *JinjaExpressionEvaluation*: in Ansible, in order to pass for example a variable as parameter to a module to be executed, its value needs to be accessed through a Jinja2 expression. This is done by delimiting the expression between double curly braces. The curly braces are not necessary in the cases in which the expression is evaluated in order to check if a certain condition is met (like for example in the "when" attribute), so in those cases the JinjaExpressionEvaluationWithoutBrackets entity is used.
  The Jinja2 expression is modelled to have 2 alternatives: being an IfExpression entity (the classic "if…then..else") or a FilteredExpression. A FilteredExpression entity can contain filters applied to the expression before being evaluated.
  A Jinja2 expression can contain various elements and operands, like "and", "or", ">", "<", "+", "-", etc. In the xtext grammar this is defined thanks to various correspondent productions with right recursions, and this finds correspondence in the metamodel. The basic element of the chain of right recursions is the ValuePassedToJinjaExpression entity, while at the top of the chain we find the FilteredExpression entity.
- *TailElement*: an expression could have a tail: basically a "." followed by a function or the identifier of an inner element, if the expression represents a dictionary. TailElement is the entity put after the "." sign. Of course, the expression could have a chain of tail elements. It's also possible, for each one of them, to access a specific element of a list (if the overall expression represents a list), so there is also the "index" attribute. The "index" attribute modelled in ParenthesisedExpression is there because the expression could be itself a list and we may want to access an element of it, so without any tail element involved.
- *ValuePassedToJinjaExpression*: This is the basic element of a Jinja2 expression.
  ValuePassedToJinjaExpression could be a value. A value, as it can intuitively be thought, is something like a number, a string, a dictionary, a list, etc.
  ValuePassedToJinjaExpression could be a SpecialVariable. This term was chosen by taking inspiration from the special variables Ansible documentation[48]. It captures the variables that contain something related to the internal state of Ansible. To make it clearer, an example could be "ansible_facts". Facts are basically properties gathered from the host on which the ansible playbook is run, like for example its IP address, and "ansible_facts" is a variable (a

[48] https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html


© Copyright Beneficiaries of the SODALITE Project

dictionary) containing those facts. Even if not in the list of the special variables, in the page of the Ansible documentation, it's reasonable to use the SpecialVariable entity also when the "item" variable needs to be used. "item" is a keyword in ansible for accessing an element of a list in a LoopOverList. It could be for example the case that we have a list of 3 elements, and we want to print each one of them. So ansible basically tells something like "print item" and this command is executed 3 times. Each time "item" will have the corresponding value of the element of the list.

Finally, ValuePassedToJinjaExpression could be a VariableReference entity, which is a reference to a variable. The following dedicated point is necessary to better explain this entity.

- *VariableReference*: it represents a reference to a variable. There are different types of variables, in the sense that there are different possible ways in which they are declared/passed to the playbook. Thus, there are different types of VariableReference entities:
    - *VariableDeclarationVariableReference*: it could be identified as a reference to the "classic" declared variable: the VariableDeclaration entity associates an identifier of the variable to a value, and VariableDeclarationVariableReference is a reference to VariableDeclaration.
    - *RegisterVariableReference*: it's a reference to RegisterVariable, an entity that models the variables that are produced in ansible after the execution of a task or a handler: after the execution, the output of it can be stored in a new variable, the registered variable.
    - *InputOperationVariableReference* and *InputInterfaceVariableReference*: They are both related to variables given in input from the TOSCA Resource Model to the playbook. The main difference between them is that the variable given in input could be defined in the "operation" section of the interface, like it is done in the Snow example[49] for "remote_server" and mysql_db_pass" (in that case we have the InputOperationVariableReference) or in the interface itself, like it's done in the HPC example[50] for "wm_public_address", "wm_username", etc. (so we have the InputInterfaceVariableReference). The main reason for having these 2 different entities, instead of just one capturing both the cases, is that in the first case the RM grammar uses the ParameterDefinition entity to define the input, while in the second case it uses the PropertyDefinition entity. Since the inputs are defined in these 2 different ways, it was reasonable to define 2 different correspondent entities for referencing them.
    - *IndexOrLoopVariableReference*: while looping, it's possible to define new variables with the index_var[51] keyword and the loop_var[52] keyword. This entity is just used for capturing both the cases.

[49] https://github.com/SODALITE-EU/ide/blob/master/dsl/org.sodalite.dsl.examples/snow_split.v2/snow_v2.rm#L19

[50] https://github.com/SODALITE-EU/ide/blob/master/dsl/org.sodalite.dsl.examples/hpc/hpc.rm#L87

[51] https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#tracking-progress-through-a-loop-with-index-var

[52] https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#defining-inner-and-outer-variable-names-with-loop-var


© Copyright Beneficiaries of the SODALITE Project

- *ValuePassed*: First, it's necessary to explain what a value in this model is. It can be a simple value (string, boolean, number, null) or a composed value (a list, a dictionary).
  The JinjaExpressionEvaluation, explained before, was introduced to model the values passed to a module to be executed, but it's not enough alone. As it can be seen in the create docker host example[53], a value passed could be a JinjaExpressionEvaluation concatenated with a string. So, we have introduced the JinjaExpressionOrString entity, that can be a JinjaExpressionEvaluation or a string. JinjaExpressionAndString, instead, is an ordered set of JinjaExpressionOrString entities. In this way, we have the JinjaExpressionAndString entity that models the concatenation of strings and jinja expressions.
  To sum up, a value passed could be a JinjaExpressionAndString or a value. Because JinjaExpressionAndString contains strings, this would lead to ambiguity in the grammar associated with this model, since both Value and JinjaExpressionAndString entities could be a string. For this reason, ValueWithoutString was introduced in the model, which is basically like Value but it can't be a string.
  For what concerns the attributes in general, like for example "delegate_facts", "async", "check_mode", etc., the value that is passed to them is not just simply ValuePassed. The point here is that, thanks to the Ansible documentation and repository[54] we know in advance what is the type expected from each attribute. If, like in the case of "check_mode"[55], the value expected is a boolean, what can actually be passed to "check_mode" is a boolean or a jinja expression (which should at run time be a boolean, but because of the strong relation between this metamodel and the xtext grammar defined in the IDE, here we are focusing rather on the compile time). So BooleanPassed is the entity that can be a Boolean or a JinjaExpressionEvaluation, and is the one passed to "check_mode". In this way, if the programmer writes something like "check_mode: 5", the IDE will raise an error message, saying that the number type isn't compatible with "check_mode". The same consideration holds for DictionaryPassed, ListPassed, NumberPassed. For what concerns String, JinjaExpressionAndString is already fine for this purpose, so that can be used instead of defining a StringPassed entity.
  This kind of approach, that aims at forcing the programmer to pass the values with the right type, wasn't used for the Parameter entity (recall: it's the one involved in the ModuleCall entity) simply because in general we don't know what is the right type of value that should be passed as a parameter to a general module. We could know it if we consider a specific module: if we select one from the list of modules[56] of the Ansible documentation, we can see the list of its parameters and for each one the expected type. If the IDE can access in some way (with some endpoints for example) a component that provides this information, then in principle it is possible to use an approach similar to the one just described for "check_mode" for the parameters of the modules. However, this metamodel is general and doesn't take into account this possibility. In general, one could, with the ModuleCall entity, "call" every possible module (even not existing in the list of modules) by providing an identifier, and could pass every parameter that he/she wants, by providing an identifier. So, of course, under these circumstances it's inevitable to let the programmer use every possible type, with the ValuePassed entity.

[53] https://github.com/SODALITE-EU/iac-management/blob/master/use-cases/snow-uc/snow-openstack/playbooks/docker/create_docker_host.yml#L67
[54] https://github.com/ansible/ansible
[55] https://github.com/ansible/ansible/blob/devel/lib/ansible/playbook/base.py#L613
[56] https://docs.ansible.com/ansible/2.9/modules/list_of_all_modules.html


© Copyright Beneficiaries of the SODALITE Project