



PIACERE

Deliverable D2.2

PIACERE DevSecOps Framework Requirements specification, architecture and integration strategy – v2

Editor(s):	Emanuele Morganti
Responsible Partner:	Hewlett Packard Enterprise - HPE
Status-Version:	Final-v1.0
Date:	01.12.2022
Distribution level (CO, PU):	PU

Project Number:	101000162
Project Title:	PIACERE

Title of Deliverable:	PIACERE DevSecOps Framework Requirements specification, architecture and integration strategy – v2
Due Date of Delivery to the EC	31.10.2022

Workpackage responsible for the Deliverable:	WP2 - PIACERE Requirements, Architecture and DevSecOps
Editor(s):	Hewlett Packard Enterprise - HPE
Contributor(s):	Aleš Černivec – XLAB Annelisa Motta – HPE Elisabetta Di Nitto – POLIMI Eliseo Villanueva Morte - Prodevelop Emanuele Morganti – HPE Eneko Osaba Icedo – Tecnalía Jesús López Lobo - Tecnalía Galia Novakova Nedeltcheva – POLIMI Iñaki Etxaniz - Tecnalía Gorka Benguria Elguezabal – Tecnalía Ismael Torres Boigues – Prodevelop Lorenzo Blasi – HPE Matija Cankar – XLAB Paweł Skrzypek – 7BULLS Radosław Piliszek – 7BULLS
Reviewer(s):	Matija Cankar (XLAB)
Approved by:	All Partners
Recommended/mandatory readers:	WP2, WP3, WP4, WP5, WP6, WP7

Abstract:	This document is the second release of the <i>PIACERE DevSecOps Framework requirements Specification, architecture and integration strategy</i> document. It contains 1) the updated list of the functional and non-functional requirements related to the PIACERE DevSecOps Framework, including its components; 2) the updated architecture of the DevSecOps framework [KR13] and the workflow; 3) the updated requirements of the DevOps infrastructure to be used in the development of PIACERE as well as the updated steps to be followed for the continuous integration of the PIACERE solution.
Keyword List:	Architecture, Integration Strategy
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the author’s views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	26.09.2022	First draft version including TOC and assignments	HPE
v0.2	25.10.2022	Updates of IaC Scan Runner. Suggestions and updates received by consortium partners	All
v0.3	15.11.2022	Document fully reviewed, ready for internal review and candidate to the final version v1.0	HPE
v0.4	25.11.2022	Revision of content after quality review by Matija Cankar (XLAB)	HPE
v0.5	29.11.2022	Second Revision of content after quality review by Matija Cankar (XLAB)	HPE
v1.0	01.12.2022	Ready for submission	TECNALIA

DRAFT

Table of contents

Executive Summary	8
1 Introduction	10
1.1 About this deliverable	10
1.2 Document structure	10
1.3 Key Results (KRs) relationship	10
2 Requirements Specification	12
2.1 Changes in v2	12
2.2 General description	12
2.3 Requirements Collection	12
2.3.1 Functional Requirements	15
2.3.2 Non-Functional Requirements	19
2.3.3 Business Requirements	19
2.3.4 Use Cases mapped on requirements	20
2.4 Requirements Summary Dashboard	25
3 PIACERE Architecture	27
3.1 Changes in v2	27
3.2 General description	27
3.3 Logical/Functional View	28
3.4 Architecture components	38
3.4.1 Integrated Development Environment - IDE (KR2)	38
3.4.2 DevOps Modelling Language – DOML/DOML-E (KR1-KR4)	40
3.4.3 Infrastructural Code Generator - ICG (KR3)	41
3.4.4 Verification Tool - VT	44
3.4.5 IaC Executor Manager – IEM (KR10)	48
3.4.6 Runtime Controller – PRC	50
3.4.7 Canary Sandbox Environment – CSE (KR8)	50
3.4.8 Infrastructure Advisor	52
3.4.9 Infrastructural Elements Catalogue (KR9)	59
3.5 PIACERE Multi-User Approach	60
3.6 PIACERE Security Approach	62
3.7 PIACERE Scenarios	63
4 Integration Strategy (KR13)	67
4.1 Changes in v2	67
4.2 Integration strategy – definitions	67
4.3 Framework components	68
4.3.1 Integration Repository	68

4.3.2	CI/CD Flow	68
4.4	Framework description DevOps Pipeline	68
4.5	Selection of integration strategy	68
5	Conclusions	70
6	References	71
APPENDIX: PIACERE Glossary		72
Changes in v2		72
Glossary structure		72
Basic Terms.....		72
The application		72
Technical Requirements (TR)		72
Non-Functional Requirements (NFR)		73
Configuration Management		73
Infrastructure Provisioning.....		73
Orchestration		73
Infrastructure as Code (IaC)		73
Infrastructure as a Service (IaaS).....		74
Target IaC Language (TlaCL)		74
Configuration Drift		74
DevOps Modelling Language (DOML)		75
Infrastructure Element (IE).....		75
PIACERE design time		75
PIACERE runtime		75
Resource Provider (RP).....		76
Execution Environment (EE).....		76
Production Execution Environment (PEE)		76
Canary Execution Environment (CEE).....		77
Components		77
Integrated Development Environment (IDE)		77
Infrastructural Code Generator (ICG).....		77
Canary Sandbox Environment (CSE).....		78
DOML & IaC Repository		79
Infrastructural Elements Catalogue (IEC)		79
Verification Tool (VT).....		80
PIACERE Runtime Controller (PRC).....		82
IaC Executor Manager (IEM)		83
Infrastructure Advisor (IA).....		83
Addenda		87

IaaS and Cloud Computing Models	87
---------------------------------------	----

List of tables

TABLE 1: CHANGES FROM PREVIOUS VERSION (D2.1)	8
TABLE 2: REQUIREMENTS/KRS	14
TABLE 3: FUNCTIONAL REQUIREMENTS	16
TABLE 4: NON-FUNCTIONAL REQUIREMENTS	19
TABLE 5: BUSINESS REQUIREMENTS	20
TABLE 6: USE CASE AND REQUIREMENTS MAPPING	20
TABLE 7: PIACERE REQUIREMENTS SUMMARY TABLE	25
TABLE 8: PIACERE DESIGN WORKFLOW	29
TABLE 9: PIACERE RUNTIME WORKFLOW	34
TABLE 10: PIACERE TEST WORKFLOW	37
TABLE 11: PIACERE MULTI-USER APPROACH	60
TABLE 12: PIACERE SCENARIOS	63
TABLE 13: TERMS AND ACRONYMS FOR INTEGRATION STRATEGY	67
TABLE 14: INTEGRATION STRATEGY EVALUATION CRITERIA	69

List of figures

FIGURE 1: PIACERE KEY RESULTS	11
FIGURE 2: KEY RESULTS RELATIONSHIP	11
FIGURE 3: REQUIREMENT’S COLLECTION WORKFLOW	13
FIGURE 4: PIACERE REQUIREMENTS SUMMARY DASHBOARD	26
FIGURE 5: PIACERE DESIGN TIME	29
FIGURE 6: KRS INVOLVED IN PIACERE DESIGN TIME	31
FIGURE 7: PIACERE RUNTIME – DEPLOYMENT ACTIVITIES	32
FIGURE 8: PIACERE RUNTIME – MONITORING ACTIVITIES	33
FIGURE 9: KRS INVOLVED IN PIACERE RUNTIME	37
FIGURE 10: IDE SEQUENCE DIAGRAM	39
FIGURE 11: INTERACTION OF THE PIACERE USER WITH DOML AND THE IDE	41
FIGURE 12: INTERNAL ICG ARCHITECTURE	42
FIGURE 13: ICG INTERNAL AND EXTERNAL BEHAVIOUR	43
FIGURE 14: INTERNAL ARCHITECTURE OF THE MODEL CHECKER	44
FIGURE 15: MODEL CHECKER INTERNAL AND EXTERNAL BEHAVIOUR	45
FIGURE 16: IAC SCAN RUNNER ARCHIVE SCAN WORKFLOW WITH PERSISTENCE AND CONFIGURATIONS	46
FIGURE 17: IAC SECURITY AND COMPONENT SECURITY INSPECTOR	48
FIGURE 18: START OF DEPLOYMENT	49
FIGURE 19: REQUEST OF THE STATUS OF A DEPLOYMENT	50
FIGURE 20: CANARY SANDBOX ENVIRONMENT PROVISIONER (CSEP)	51
FIGURE 21: IOP IN RUN TIME	53
FIGURE 22: IOP IN DESIGN TIME	53
FIGURE 23: MONITORING	55
FIGURE 24: MONITORING SYSTEM	56
FIGURE 25: SELF-LEARNING (PERFORMANCE)	57
FIGURE 26: SECURITY SELF-LEARNING	58
FIGURE 27: SELF-HEALING	59
FIGURE 28: INFRASTRUCTURE ELEMENTS CATALOGUE	60
FIGURE 29: STATUS OF CLOUD COMPUTING MODELS (SOURCE: H-CLOUD)	88

Terms and abbreviations

Amazon EC2	Amazon Elastic Compute Cloud
API	Application Programming Interface
AWS	Amazon Web Services
CEE	Canary Production Execution Environment
CRP	Canary Resource Provider
CSE	Canary Sandbox Environment
CSEM	Canary Sandbox Environment Mocklord
CSPE	Canary Sandbox Environment Provisioner
CSI	Component Security Inspector
CSP	Cloud Service Provider
DevOps	Development and Operations
DoA	Description of Action
DOML	DevOps Modelling Language
DOML-E	DevOps Modelling Language -Extensions
EC	European Commission
EE	Execution Environment
FR	Functional Requirement
GA	Grant Agreement to the project
GUI	Graphical User Interface
IA	Infrastructure Advisor
IaC	Infrastructure as Code
ICG	Infrastructure Code Generator
IDE	Integrated Development Environment
IEC	Infrastructural Elements Catalogue
IEM	IaC Executor Manager
IOP	IaC Optimizer Platform
KPI	Key Performance Indicator
KR	Key Result
MC	Model Checker
MDE	Model-Driven Engineering
PEE	Production Execution Environment
PRC	Piacere Runtime Controller
PRP	Production Resource Provider
REQ	Requirement
RP	Resource Provider
SW	Software
TR	Technical Requirement
UC	Use Case
VT	Verification Tool
WP	Work Package
Y1	Year 1
Y2	Year 2
Y3	Year 3

Executive Summary

This deliverable is the version 2 release of following WP2 tasks outcome:

- Task 2.1-Requirement’s specification, with the aim to define the PIACERE functional and non-functional requirements as well to identify requirements for the use cases identified in WP7 (Use Case Validation);
- Task 2.2-PIACERE Architecture definition with the aim to describe how PIACERE components interact with each other;
- Task 2.3-PIACERE DevSecOps delivery strategy and continuous integration with the aim to integrate all PIACERE components (KR1-KR12).

This document is therefore an update at M24 of the deliverable [D2.1, PIACERE DevSecOps Framework requirements Specification, architecture and integration strategy-v1](#) released at M12. As it reflects the architecture of the final PIACERE solution framework, the document reports about updates in the second year of the project, but it includes also the sections and parts that have remained unchanged to maintain consistency and completeness.

The main changes in version 2 are related to **Requirements Specification** and **PIACERE Architecture Definition** topics (**section 2 and section 3 of this document**) as reported in the table below:

Table 1: Changes from previous version (D2.1)

Topic	Description of changes from previous version (D2.1)
Requirements Specification (Task 2.1)	<ul style="list-style-type: none"> • The requirement collection process has been improved to allow re-discussion for specific requirements considering feedback from integration and UC validation. • The relationship between requirements and KRs has been reviewed and in some cases updated after in-depth analysis. • New requirements have been proposed and accepted. • The list of requirements for the development of PIACERE components has been updated and completed, except for minor adjustments resulting from Piacere component integration session and UC validation still on going.
Architecture Definition (Task 2.2)	<ul style="list-style-type: none"> • The general architecture workflows of design and runtime have been improved and updated in the communication mechanisms between some KRs, identifying requests, responses and user interactions. • Some components can be involved in both design and runtime phases: ICG (KR3) is mainly involved in the design phase, but it can also be invoked in the runtime phase; similarly, IOP (KR9) is mainly involved in the runtime phase, but it can be invoked in the design phase. The sequence diagrams of ICG, IOP and IDE have been changed accordingly. • Self-healing mechanism and related sequence diagrams have been updated: if the self-learning or the monitoring component detects a failure or potential failure, the self-healing component receives an alerting message. The self-healing component categorizes the incidence to start the appropriate workflows (redeploy, scale, quarantine) calling the PRC. • The IaC Scan Runner component has been added to manage the security checks.

	<ul style="list-style-type: none">• PRC remains the sole contact between the design time (IDE) and runtime tools: the sequence diagrams of IEM and IDE have been changed accordingly.• The sequence diagram of Infrastructural Elements Catalogue has been updated to point out its role in design time and runtime phases.• An overview to the multi-user approach has been presented.• Some scenarios of the using of PIACERE framework from the user perspective have been described.
Integration Strategy (Task 2.3)	No changes respect version 1 on the strategy to follow for the continuous integration of the PIACERE solution.
PIACERE Glossary	The glossary, which includes the most common terms used in PIACERE with a high-level description of all components, has been updated including the IaC Scan Runner component that acts as KR6-KR7 executor.

DRAFT

1 Introduction

This deliverable provides an analysis on the different architectural aspects that describe how PIACERE framework works and what are the main building blocks of the solution. It reports about updates in the outcomes of requirements specification and PIACERE architecture definition tasks in the second year of the project, but it also includes the sections and parts that have remained unchanged to maintain consistency and completeness. In that sense it can be considered as a *rolling* deliverable.

1.1 About this deliverable

The deliverable will serve as an *architectural document* for the other work packages of PIACERE project that are involved in developing blocks of the PIACERE solution.

It contains all functional and non-functional requirements, selected from different sources, to develop each component and to integrate each other generating the DevSecOps PIACERE Framework, explaining also the approach used to propose and select the requirements.

It describes the outcome of architectural analysis work of PIACERE framework, showing the workflow and interaction between components and the multi-user approach.

Finally, it presents the strategy and steps to be followed for the continuous integration of the PIACERE solution.

1.2 Document structure

The rest of this document is structured as follows:

- Section 1 presents an overall description of the deliverable and its main goal is provided.
- Section 2 focuses on the outcome, at M24, of the analysis of requirements related to the development of PIACERE platform and the process used to select them.
- Section 3 presents the description of the final PIACERE architectural design choices, observed from different perspectives, highlighting the workflow with internal and external communication mechanisms details and the multi-user approach.
- Section 4 presents the strategy to follow for the continuous integration of the PIACERE solution.
- Section 5 presents a summary of discernments achieved through this deliverable and draws the conclusions.
- Section 6 presents any relevant additional documentation as citations.
- APPENDIX: PIACERE Glossary provides a glossary of the terms used within PIACERE to effectively unify the vocabularies and describes the main components that involve the PIACERE architecture.

1.3 Key Results (KRs) relationship

The main objective of this deliverable is to provide requirements specification (initial and updated in last year) for the different Key Results (KRs) under development in PIACERE project and to describe what are the main building blocks of the PIACERE framework. The following two figures Figure 1 and Figure 2 represent the different KRs and the relationship between KRs.

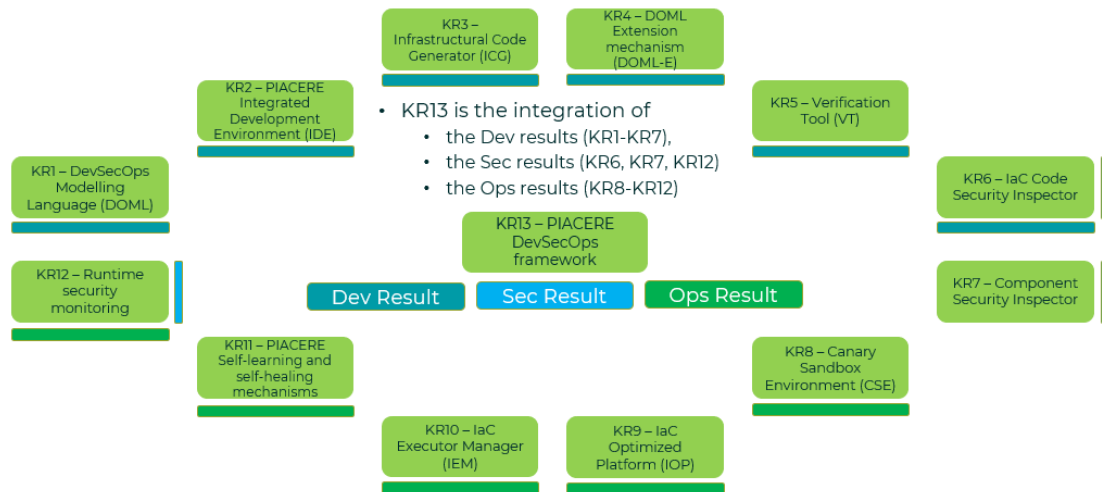


Figure 1: PIACERE Key Results

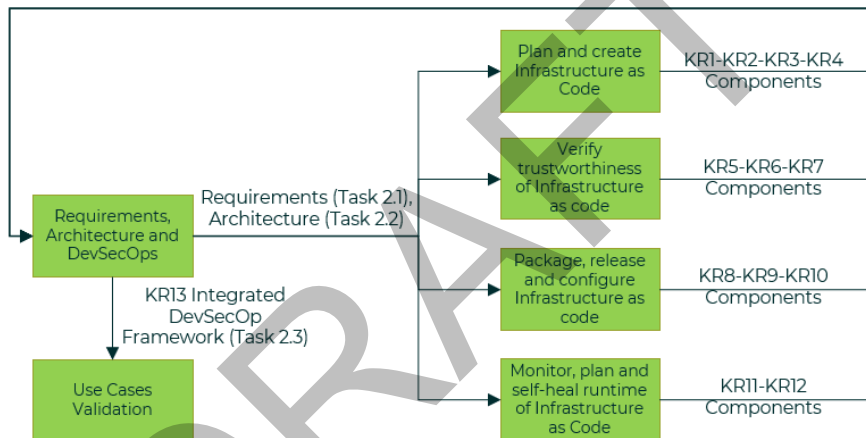


Figure 2: Key Results relationship

2 Requirements Specification

This section describes the process to analyse and define the PIACERE requirements for the development of PIACERE components (KR1-KR13).

2.1 Changes in v2

This section reports the updates in the outcome of requirement's specification task in the second year of the project. The requirement collection process has been improved to allow re-discussion for specific requirements considering feedback from integration and UC validation. The relationship between requirements and KRs has been reviewed in Table 2 and in some cases updated after in-depth analysis on the impacts. Requirements in Table 3 and Table 4 have been updated with new requirements and removing the discarded ones after the review process. The mapping between requirements, KRs and UC has been updated with the new requirements in Table 6. The requirements summary dashboard has been updated as well (Table 7, Figure 4). The details of changes are reported in the sub-section below.

2.2 General description

The purpose of this section is to list the requirements collected for implementing the PIACERE solution, grouped by typology [1]:

- **Functional requirements** are presented as lists of features or services that the system has to provide according to the assigned priority. They also describe the behaviour of the system in the face of particular inputs and how it should react in certain situations.
- **Non-Functional requirements** represent system-related constraints and properties, such as time constraints, constraints on the development process and on the standards to be adopted. Non-functional requirements are not just about the software system being developed; some may constrain the process used to develop the system (e.g., performance, usability).
- **Business requirements** provide the scope, business needs or issues that need to be addressed through specific activities. These requirements provide the information to ensure that the PIACERE project achieves the identified objectives.

Regarding requirements there are two different perspectives: in the Requirement Specification section we specify the requirements to implement the PIACERE solution (section 2.3.1). On the other hand, the PIACERE solution has to offer to end-users the ability to express requirements that are related to the system they want to run through the PIACERE solution. This topic is more detailed in the APPENDIX PIACERE Glossary sections Technical Requirements (TR), Non-Functional Requirements (NFR).

2.3 Requirements Collection

To achieve the purpose of analysis and definition of the PIACERE requirements, an iterative process that involves all partners has been set up at the beginning of the project. It has been improved after Y1 to allow re-discussion for specific requirements during the KR development stage. Below it is described the updated workflow:

- Each new requirement is proposed with adding a new row in a shared spreadsheet specifying the following fields:
 - **Description** - short description of requirement
 - **Type** - possible values: functional, non-functional, business
 - **Complexity** - possible values: low, medium, high, N.A.
 - **Involved KR and Involved WP(s)/task(s)** - list of KRs and involved task
 - **Source** - possible values: DoA, Use Case, Other

- **Status** - proposed
 - **Priority** [2] - possible values: must have, should have, could have, won't have
 - **Timeline** - possible values according to priority: Y1, Y2, Y3
- Each requirement collected is analysed and discussed according to the workflow described in Figure 3.
 - During the analysis and discussion of the requirements, the relationships between requirements, KR's and UC's are checked and updated if necessary. In addition, the priorities and the timeline are reviewed.
 - When the workflow has been completed the status of each requirement can be 'duplicate', 'discarded' or 'accepted'. The status under 'discussion' or 'proposed' means that the workflow is still on-going for that requirement.
 - When a further deep discussion is needed for an 'accepted' or 'discarded' requirement the field 'Re-discuss' set at 'yes' can restart the workflow.

To identify the relationship between the requirements and the supported UC's, the mapping between requirements, KR's and UC's is achieved by adding the following values for each requirement in the columns UC1, UC2 and UC3: *UC Priority ; Impact ; Version*.

Column UC1, UC2 and UC3 refer respectively to Slovenian Ministry of Public Administration, Critical Maritime Infrastructures and Public Safety on IoT in 5G use cases.

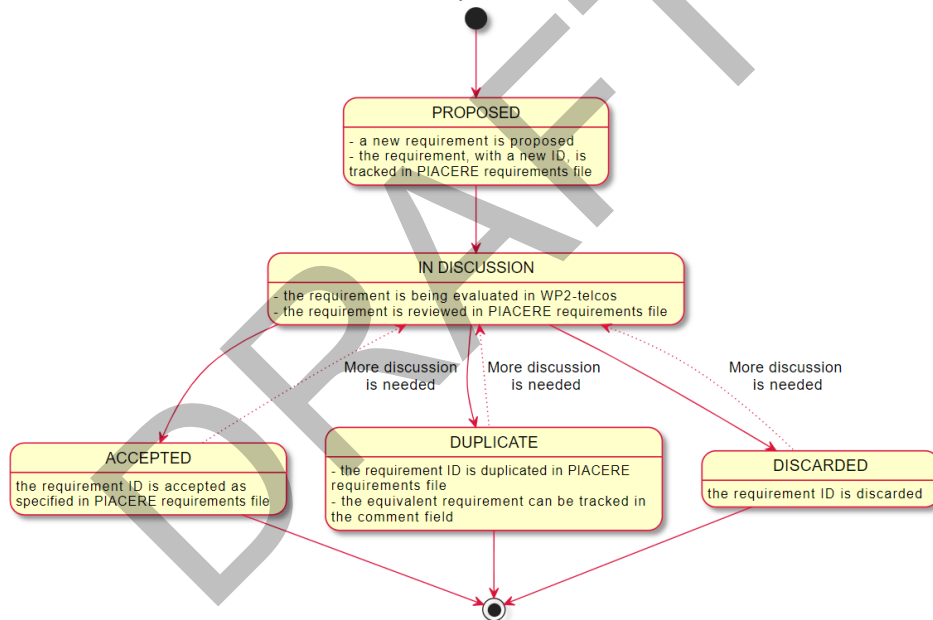


Figure 3: Requirement's collection workflow

In this document only requirements without the 'discarded' or 'duplicate' status are presented. As the task 2.1 related to requirement's specification is closed, the list of the requirements is the final version. However, minor adjustments to the requirements might be necessary during the last stage of PIACERE tools development and integration.

In Table 2 it is presented the updated mapping between requirements (REQ) and PIACERE's Key Results (KR1/KR13) with the planned *Timeline* estimated to achieve each requirement. In this table, the relationship between requirements and KR's has been reviewed in Y2 and in some cases updated after in-depth analysis on the requirements and impacts.

For each row, the 'x' in a cell specifies the key result (one or more than one) to which that requirement refers. In the last row, a **grand total count** is added to gain visibility of distribution of REQ's towards KR's. The last column (Status), shown the implementation status of the

requirement. The possible values for this column are: R - realized ; tbv - to be validated ; tbm - to be made.

The table shows that 58% of requirements are related to KR's focusing to the design, planning and verify the trustworthiness of IaC (PIACERE design time), indication already obtained in the studies conducted for the first version of this document. Almost all requirements have been implemented in Y1 and Y2 to allow the validation process of the PIACERE Use Cases. The status of the achievement for each requirement is monitored by involved KR's owners, adjusting the *Timeline* if needed.

More details regarding the relationship between Requirements, KR's and work packages are highlighted in the dashboard section 2.4.

Table 2: Requirements/KR's

REQ ID	KR 1	KR 2	KR 3	KR 4	KR 5	KR 6	KR 7	KR 8	KR 9	KR 10	KR 11	KR 12	KR 13	Timeline	Status
REQ01	x			x										Y1	R
REQ03									x					Y1	R
REQ04									x					Y2	tbv
REQ10													x	Y2	tbv
REQ11											x			Y1	R
REQ12										x				Y2	tbv
REQ14												x		Y1	R
REQ15												x		Y2	tbv
REQ16											x	x		Y2	tbv
REQ17											x	x		Y1	R
REQ18												x		Y1	R
REQ19												x		Y2	tbv
REQ21												x		Y2	tbv
REQ23							x							Y1	R
REQ24						x	x							Y1	R
REQ25	x													Y1	R
REQ26	x													Y1	R
REQ27	x			x										Y1	R
REQ28	x	x												Y1	R
REQ29	x			x										Y1	R
REQ30	x			x										Y2	tbv
REQ31			x											Y2	tbv
REQ33								x						Y1	R
REQ34								x						Y1	R
REQ36	x			x										Y2	tbv
REQ37								x						Y2	tbv
REQ38								x						Y1	R
REQ39								x						Y2	tbv
REQ40		x												Y1	R
REQ41		x												Y2	tbv
REQ42		x												Y1	R
REQ43		x												Y1	R
REQ44		x												Y2	tbv
REQ46									x		x			Y1	R
REQ47											x			Y1	R
REQ48											x			Y2	tbv
REQ50											x	x		Y1	R

REQ ID	KR 1	KR 2	KR 3	KR 4	KR 5	KR 6	KR 7	KR 8	KR 9	KR 10	KR 11	KR 12	KR 13	Timeline	Status
REQ51											x	x		Y1	R
REQ52											x			Y1	R
REQ55										x				Y2	tbv
REQ57	x													Y2	tbv
REQ58	x													Y2	tbv
REQ59	x													Y2	tbv
REQ60	x													Y1	R
REQ61	x													Y1	R
REQ62	x	x												Y1	R
REQ63	x													Y1	R
REQ64		x												Y2	tbv
REQ65						x	x							Y1	R
REQ66						x								Y1	R
REQ67							x							Y1	R
REQ70	x													Y1	R
REQ72											x			Y2	tbv
REQ76	x	x												Y1	R
REQ77			x											Y1	R
REQ81										x				Y1	R
REQ82										x				Y2	tbv
REQ83										x				Y2	tbv
REQ84										x				Y2	tbv
REQ85										x				Y2	tbv
REQ87										x				Y1	R
REQ88													x	Y2	tbv
REQ92											x			Y1	R
REQ93											x			Y1	R
REQ94											x			Y2	tbv
REQ95					x									Y1	R
REQ96			x											Y1	R
REQ97											x			Y2	tbv
REQ98									x					Y2	tbv
REQ99		x												Y1	R
REQ100			x											Y1	R
REQ101		x												Y2	tbv
REQ103					x									Y1	R
REQ104					x									Y1	R
REQ105					x									Y1	R
REQ106						x	x							Y3	tbm
REQ107						x	x							Y3	tbm
REQ108						x	x							Y3	tbm
REQ109						x	x							Y3	tbm
REQ110			x											Y3	tbm
Grand total	17	11	5	5	4	7	8	5	4	8	14	9	2		

2.3.1 Functional Requirements

In Table 3 it is presented the updated list of functional requirements without the 'discarded' and 'duplicate' status to be considered for the development of the involved KR. Respect the previous version REQ101 has been added for KR2; REQ100 for KR3, from REQ103 to REQ105 for

KR5, REQ106 and REQ109 for KR6, KR7; conversely REQ68, REQ69, REQ71, REQ78, REQ79, REQ80 and REQ89 have been removed as they were discarded.

Table 3: Functional requirements

REQ ID	Description	Priority	Timeline	Involved KRs
REQ01	The DOML must be able to model infrastructural elements.	MUST HAVE	Y1	KR1, KR4
REQ03	IOP will include a catalogue of infrastructural elements - e.g., node computation, networks, cloud services like IaaS, PaaS, SaaS - classifiable by a set of constraints - e.g., memory, disk. This catalogue of infrastructural elements should be clearly defined, including possible restrictions and dynamic variations. These infrastructural elements will be transformed as optimization variables, and they will be intelligently treated by the optimization algorithm seeking to find the best configuration deployment.	MUST HAVE	Y1	KR9
REQ04	Provide the means for the IOP to properly consume all the data related with the catalogue of infrastructural elements status, as well as their characteristics and possible variations. Special mention shall be done here to the values monitored by the self-learning algorithm / monitoring component. This module shall provide real measures regarding the infrastructural elements in order to update their characteristics.	MUST HAVE	Y2	KR9
REQ12	The IEM shall allow redeployment and reconfiguration, both full and partial, as allowed by the used IaC technology.	MUST HAVE	Y2	KR10
REQ14	Runtime security monitoring must provide monitoring data from the infrastructure's hosts with regard to security metrics.	MUST HAVE	Y1	KR12
REQ15	Runtime security monitoring could provide monitoring data from the application layer (infrastructure's guest) with regard to security metrics.	COULD HAVE	Y2	KR12
REQ16	Runtime security monitoring should contribute to mitigation actions taken when considering plans and strategies for runtime self-healing actions.	SHOULD HAVE	Y2	KR11, KR12
REQ18	Runtime security monitoring must be able to detect different types of metrics in run-time: integrity of IaC configuration, potential attacks to the infrastructure, IaC security issues (known CVEs of the environment).	SHOULD HAVE	Y1	KR12
REQ19	Runtime security monitoring and alarm system (self-learning) integration must be implemented.	MUST HAVE	Y2	KR12
REQ21	Runtime security monitoring and Runtime monitoring infrastructure should be integrated with minimal extensions.	SHOULD HAVE	Y2	KR12
REQ23	IaC Code Security Inspector must analyse IaC code with regard to security issues of the modules used in the IaC.	MUST HAVE	Y1	KR7
REQ24	Security Components Inspector must analyse and rank components and their dependencies used in the IaC.	MUST HAVE	Y1	KR6, KR7
REQ25	DOML should support the modelling of security rules (e.g., by type TCP/UDP, and ingress/egress port definition).	MUST HAVE	Y1	KR1
REQ26	DOML should support the modelling of security groups (containers for security rules).	MUST HAVE	Y1	KR1
REQ27	DOML should support the modelling, provisioning, configuration and usage container engine execution technologies (e.g., docker-host).	SHOULD HAVE	Y1	KR1, KR4
REQ28	DOML should support the modelling of containerized application deployment (e.g., pull/run/restart/stop docker containers).	MUST HAVE	Y1	KR1, KR2
REQ29	DOML should support the modelling of VM provisioning for different platforms such as (OpenStack, AWS) for canary and production environments.	MUST HAVE	Y1	KR1, KR4
REQ31	ICG should provide verifiable and executable IaC generated from DOML for selected IaC languages (e.g., TOSCA/Ansible/Terraform).	MUST HAVE	Y2	KR3

REQ ID	Description	Priority	Timeline	Involved KR8
REQ33	CSE to provide a viable alternative target for IaC executors to run against, i.e., usable by the IaC Executor Manager (IEM).	MUST HAVE	Y1	KR8
REQ34	CSE to keep track of and allow querying of the deployment state to allow comparison against the expected one.	MUST HAVE	Y1	KR8
REQ36	DOML to enable writing infrastructure tests.	MUST HAVE	Y2	KR1, KR4
REQ38	CSE to have a "real" mode where resources are really provided and can be used for configuration and other further steps.	MUST HAVE	Y1	KR8
REQ39	CSE to enable extensibility (documented way): adding new mocked services, adding new "real" deployments.	SHOULD HAVE	Y2	KR8
REQ40	The IDE should provide a visual diagram functionality to visualise the different assets defined through the DOML and DOML Extensions.	MUST HAVE	Y1	KR2
REQ41	The IDE should be extensible through the plugin mechanism. Not only to support PIACERE assets (ICG, VT) but also for third party collaborators.	MUST HAVE	Y2	KR2
REQ43	The IDE should be easily updatable to newer software versions.	MUST HAVE	Y1	KR2
REQ44	The IDE could provide an import mechanism to automatically fulfil partial DOML.	COULD HAVE	Y2	KR2
REQ46	The monitoring component shall gather metrics from the instances of the infrastructural elements at run time. These metrics need to be related to the TR and accessible for the IOP (through the dynamic part of the infrastructural catalogue).	MUST HAVE	Y1	KR9, KR11
REQ47	The monitoring component shall include the needed elements in the stack to monitor the infrastructural elements.	MUST HAVE	Y1	KR11
REQ48	The monitoring component shall transform the real time values into the correct format/type/nature for the self-learning component.	MUST HAVE	Y2	KR11
REQ50	The monitoring component shall monitor the metrics associated with the defined measurable NFRs (e.g., performance, availability, and security through the runtime security monitoring).	MUST HAVE	Y1	KR11, KR12
REQ51	The self-learning component shall ensure that the conditions are met (compliance with respect to SLO) and that a failure or a non-compliance of a NFRs is not likely to occur. This implies the compliance of a predefined set of non-functional requirements (e.g., performance).	MUST HAVE	Y1	KR11, KR12
REQ52	Self-learning shall consume the data monitored and store it in a time-series database to create discriminative complex statistical variables and train a predictor which will learn potential failure patterns in order to prevent the system from falling into an NFR violation situation.	MUST HAVE	Y1	KR11
REQ55	The IEM will log the whole IaC execution run, making metadata and metrics (time it took to run) about the creation of resources available to the rest of the PIACERE components.	MUST HAVE	Y2	KR10
REQ57	It is desirable to enable both forward and backward translations from DOML to IaC and vice versa.	SHOULD HAVE	Y2	KR1
REQ58	DOML should offer the modelling abstractions to define the outcomes of the IOP.	MUST HAVE	Y2	KR1
REQ59	The DOML should allow users to define rules and constraints for redeployment, reconfiguration and other mitigation actions.	MUST HAVE	Y2	KR1
REQ60	DOML should support the modelling of security metrics both at the level of infrastructure and application.	MUST HAVE	Y1	KR1

REQ ID	Description	Priority	Timeline	Involved KR
REQ61	DOML must support the modelling of TRs and of SLOs.	MUST HAVE	Y1	KR1
REQ62	DOML must support different views.	SHOULD HAVE	Y1	KR1, KR2
REQ63	DOML must be unambiguous.	MUST HAVE	Y1	KR1
REQ65	laC Security Inspector and Component Security Inspector should hide specificities and technicalities of the current solutions in an integrated IDE.	MUST HAVE	Y1	KR6, KR7
REQ66	laC Code security inspector must provide an interface (CLI or REST API) to integrate with other tools or CI/CD workflows.	MUST HAVE	Y1	KR6
REQ67	laC Component security inspector must provide an interface (CLI or REST API) to integrate with other tools or CI/CD workflows.	MUST HAVE	Y1	KR7
REQ70	The DOML should allow users to state correctness properties in a suitable sub-language (possibly Formal Logic).	MUST HAVE	Y1	KR1
REQ72	Verification Tool must verify the completeness of the laC generated by ICG.	SHOULD HAVE	Y2	KR11
REQ76	The runtime monitoring component should provide an UI for the end users to see the monitored resources and the corresponding metrics/TRs in real time.	SHOULD HAVE	Y1	KR1, KR2
REQ77	DOML should allow the user to model each of the four considered DevOps activities (Provisioning, Configuration, Deployment, Orchestration).	SHOULD HAVE	Y1	KR3
REQ81	IEM should be able to execute laC generated by ICG for selected laC languages (e.g., TOSCA/Ansible/Terraform)	MUST HAVE	Y1	KR10
REQ82	IEM shall register the status of past and present executions and enable an appropriate way to query it.	MUST HAVE	Y2	KR10
REQ83	IEM should be able to communicate with the relevant actors (orchestrators, infrastructural elements) in a secure way.	MUST HAVE	Y2	KR10
REQ84	IEM should be able to utilize the required credentials in a secure way.	MUST HAVE	Y2	KR10
REQ85	IEM should be able to clean up the resources being allocated.	MUST HAVE	Y2	KR10
REQ87	IEM shall work against the production environment and the canary environment.	MUST HAVE	Y1	KR10
REQ92	Self-healing component shall receive notifications from the self-learning.	MUST HAVE	Y1	KR11
REQ93	Self-healing component shall classify the events received from the self-learning and derive corrective actions.	MUST HAVE	Y1	KR11
REQ94	Self-healing component shall inform the run-time controller about the different components to orchestrate (the workflow to be executed).	MUST HAVE	Y2	KR11
REQ95	VT tools (model checker) must be able read DOML language.	MUST HAVE	Y1	KR5
REQ96	ICG must be able read DOML language.	MUST HAVE	Y1	KR3
REQ97	The Self-Healing components provide feedback on the DOML code, without doing automatic writes. The end user can choose to accept or not the feedback received. The current planned implementation will send a modified DOML to PRC and PRC will communicate it to the user.	MUST HAVE	Y2	KR11
REQ98	The IOP components provide feedback on the DOML code, without doing automatic writes. The end user can choose to accept or not the feedback received.	MUST HAVE	Y2	KR9

REQ ID	Description	Priority	Timeline	Involved KRs
REQ99	IDE to integrate with both local and remote Git repositories.	MUST HAVE	Y1	KR2
REQ100	ICG should generate IaC code that supports different cloud platforms.	MUST HAVE	Y1	KR3
REQ101	IDE should allow to create and edit a graphical DOML model. Possibly starting from a palette of supported components that can be drag&drop in the graphical model.	SHOULD HAVE	Y2	KR2
REQ103	Verification Tool (model checker) must verify the structural consistency of the DOML models.	MUST HAVE	Y1	KR5
REQ104	Verification Tool (model checker) must verify the correctness of DOML models, with respect to some correctness properties provided in DOML.	MUST HAVE	Y1	KR5
REQ105	Verification Tool (model checker) must verify the completeness of DOML models.	MUST HAVE	Y1	KR5
REQ106	Organization of scan results with respect to the scan outcome. Aggregate the results of distinct scan tools in a form of unified results summary that makes distinction of the result into the following cases: passed – scan is performed without any issues detected.	MUST HAVE	Y3	KR6, KR7
REQ109	Scan configuration management. There should be a possibility to maintain multiple scans concurrently for multiple users.	SHOULD HAVE	Y3	KR6, KR7

2.3.2 Non-Functional Requirements

In Table 4 it is presented the list of non-functional requirements without the ‘discarded’ and ‘duplicate’ status to be considered for the development of the involved KRs. In respect to the previous version the REQ110 has been added for KR3, and REQ107 and REQ108 for KR6, KR7.

Table 4: Non-Functional requirements

REQ ID	Description	Priority	Timeline	Involved KRs
REQ10	The communication within the different components of the architecture should be done in a secure way (e.g., https, Keycloak).	MUST HAVE	Y2	KR13
REQ11	The learning algorithm (anomaly and drift) should be executed as fast as possible as it should provide an outcome before more data arrives.	MUST HAVE	Y1	KR11
REQ17	Deployment of runtime security monitoring should happen seamlessly or with minimal effort and configuration required by the user.	MUST HAVE	Y1	KR11, KR12
REQ30	DOML should enable support for policy definition constraints for QoS/TR requirements.	MUST HAVE	Y2	KR1, KR4
REQ37	CSE to have a simulated mode limited to provisioning.	MUST HAVE	Y2	KR8
REQ42	The IDE should be implemented using open-source software.	SHOULD HAVE	Y1	KR2
REQ88	PIACERE framework should be usable by a team of people collaborating in the development of the same IaC.	MUST HAVE	Y2	KR13
REQ107	Improvement of scan response time. Compatibility matrix is adopted containing the list of available scans for various file types detected within IaC archive. Therefore, the execution of non-compatible scans for submitted IaC archive should be avoided.	SHOULD HAVE	Y3	KR6, KR7
REQ108	Management and persistence of scan results after the scan process is finished. Scan results are persisted into database for limited period (14 days), so users can look on their previous scan tasks and the achieved outcomes.	MUST HAVE	Y3	KR6, KR7
REQ110	ICG should provide enough extensibility to: comply with the DOML extension mechanism; be capable of integrating new IaC languages.	SHOULD HAVE	Y3	KR3

2.3.3 Business Requirements

In Table 5 is presented the list of business requirements without the ‘discarded’ and ‘duplicate’ status to be considered for the development of the involved KRs. This list remains the same in respect to the version in the previous deliverable (D2.1).

Table 5: Business requirements

REQ ID	Description	Priority	Timeline	Involved KRs
REQ64	The IDE should provide a text-based representation of DOML to ease version control.	SHOULD HAVE	Y2	KR2

2.3.4 Use Cases mapped on requirements

To enable the relationship between KRs and corresponding Use cases (UC) requirements, the document reports the updated mapping between requirements, KRs and UCs. In respect to the previous version, the column KRs in Table 6 has been added. Other parts of the section remain the same as in D2.1.

In the requirements are mapped on the following three Use Cases:

- UC1: Slovenian Ministry of Public Administration
- UC2: Critical Maritime Infrastructures
- UC3: Public Safety on IoT in 5G

The mapping between requirements, KRs and UC in Table 6 has been updated with the new requirements. For each row in the columns UC1, UC2 and UC3 are reported the UC Priority, Impact and Version information according requirement collection process described in 2.3 section. Below the possible values for UC Priority, Impact and Version:

- *UC Priority – Requirement priority for the use case – possible values MUST, DESIRABLE.*
- *Impact – how requirement affects the UC – possible values: FULL, PARTIAL.*
- *Version – a version of Use Case application related to the year of release - possible values: V1-Y1/2, V2-Y2/3.*

Table 6: Use Case and requirements mapping

REQ ID	KRs	Description	UC1	UC2	UC3
REQ01	KR1, KR4	The DOML must be able to model infrastructural elements.	Must have; Full; V1-Y1.	Must have; Full; V1-Y2.	Must have; Full; V1-Y2.
REQ03	KR9	IOP will include a catalogue of infrastructural elements - e.g., node computation, networks, cloud services like IaaS, PaaS, SaaS - classifiable by a set of constraints - e.g., memory, disk. This catalogue of infrastructural elements should be clearly defined, including possible restrictions and dynamic variations. These infrastructural elements will be transformed as optimization variables, and they will be intelligently treated by the optimization algorithm seeking to find the best configuration deployment.	Must have; Full; V1-Y1.	Must have; Full; V1-Y2 (Lightweight testing).	Partially validated (not optimization) V1-Y2.
REQ04	KR9	Provide the means for the IOP to properly consume all the data related with the catalogue of infrastructural elements status, as well as their characteristics and possible variations. Special mention shall be done here to the values monitored by the self-learning algorithm / monitoring component. This module shall provide real measures regarding the infrastructural elements in order to update their characteristics.	Must have; Full; V1-Y2.	Desirable; Partial; V2-Y3.	Desirable; Partially validated V2-Y3.
REQ10	KR13	The communication within the different components of the architecture should be done in a secure way (e.g., https, Keycloak).	Must have; Full; V1-Y2.	Affects (EDI, ENS - critical infrastructures).	Must have; Full; V1-Y2.
REQ11	KR11	The learning algorithm (anomaly and drift) should be executed as fast as possible as it should provide an outcome before more data arrives.	Not validated in the UC.	Affects (probabilistic algorithms can be set to execute	Not validated in the UC.

REQ ID	KRs	Description	UC1	UC2	UC3
				up to a pre-set time limit or with multiple restarts).	
REQ12	KR10	The IEM shall allow redeployment and reconfiguration, both full and partial, as allowed by the used IaC technology.	Must have; Full; V1-Y2.	Desirable (CI/CD pipeline).	Must have; Full; V1-Y2.
REQ14	KR12	Runtime security monitoring must provide monitoring data from the infrastructure's hosts with regard to security metrics.	Must have; Full; V1-Y1.	Affects (vendor-supplied - critical infrastructures).	Not validated in the UC.
REQ15	KR12	Runtime security monitoring could provide monitoring data from the application layer (infrastructure's guest) with regard to security metrics.	Could have; Full; V1-Y2.	Affects (Desirable for full integration with vendor's toolset).	Not validated in the UC.
REQ16	KR11, KR12	Runtime security monitoring should contribute to mitigation actions taken when considering plans and strategies for runtime self-healing actions.	Should have; Full; V1-Y2.	Affects (vendor-supplied).	Not validated in the UC.
REQ17	KR11, KR12	Deployment of runtime security monitoring should happen seamlessly or with minimal effort and configuration required by the user.	Must have; Full; V1-Y1	Affects (Desirable: vendor-supplied).	Not validated in the UC.
REQ18	KR12	Runtime security monitoring must be able to detect different types of metrics in run-time: integrity of IaC configuration, potential attacks to the infrastructure, IaC security issues (known CVEs of the environment).	Should have; Full; V1-Y1.	Affects (Desirable: vendor-supplied).	Not validated in the UC.
REQ19	KR12	Runtime security monitoring and alarm system (self-learning) integration must be implemented.	Must have; Full; V1-Y2.	Affects (vendor-supplied or ad-hoc solution).	Not validated in the UC.
REQ21	KR12	Runtime security monitoring and Runtime monitoring infrastructure should be integrated with minimal extensions.	Should have; Full; V1-Y2.	Affects.	Not validated in the UC.
REQ23	KR7	IaC Code Security Inspector must analyse IaC code with regard to security issues of the modules used in the IaC.	Must have; Full; V1-Y1.	Desirable.	Must have; Full; V1-Y2.
REQ24	KR6, KR7	Security Components Inspector must analyse and rank components and their dependencies used in the IaC.	Must have; Full; V1-Y1.	Desirable.	Must have; Full; V1-Y2.
REQ25	KR1	DOML should support the modelling of security rules (e.g., by type TCP/UDP, and ingress/egress port definition).	Must have; Full; V1-Y1.	Affects.	Must have; Full; V1-Y2.
REQ26	KR1	DOML should support the modelling of security groups (containers for security rules).	Must have; Full; V1-Y1.	Affects.	Must have; Full; V1-Y2.
REQ27	KR1, KR4	DOML should support the modelling, provisioning, configuration and usage container engine execution technologies (e.g., docker-host).	Should have; Full; V1-Y1.	Desirable.	Must have; Full; V1-Y1.
REQ28	KR1, KR2	DOML should support the modelling of containerized application deployment (e.g., pull/run/restart/stop docker containers).	Must have; Full; V1-Y1.	Affects.	Must have; Full; V1-Y1.
REQ29	KR1, KR4	DOML should support the modelling of VM provisioning for different platforms such as (OpenStack, AWS) for canary and production environments.	Must have; Full; V1-Y1.	Affects (vendor-supplied).	Must have; Full; V1-Y1.
REQ30	KR1, KR4	DOML should enable support for policy definition constraints for QoS/TR.	Must have; Full; V1-Y2.	Affects.	Must have; Full; V1-Y2.
REQ31	KR3	ICG should provide verifiable and executable IaC generated from DOML for selected IaC languages (e.g., TOSCA/Ansible/Terraform).	Must have; Full; V1-Y2 (at	Affects (vendor-supplied	Must have; Full; V1-Y1.

REQ ID	KRs	Description	UC1	UC2	UC3
			least Ansible).	or ad-hoc solution).	
REQ33	KR8	CSE to provide a viable alternative target for IaC executors to run against, i.e., usable by the IaC Executor Manager (IEM).	Must have; Full; V1-Y1.	Affects (redundancy desirable for resiliency and fault-prevention).	Not validated in the UC.
REQ34	KR8	CSE to keep track of and allow querying of the deployment state to allow comparison against the expected one.	Must have; Full; V1-Y1.	Affects (necessary).	Not validated in the UC.
REQ36	KR1, KR4	DOML to enable writing infrastructure tests.	Must have; Full; V1-Y1.	Affects.	Must have; Full; V1-Y2.
REQ37	KR8	CSE to have a simulated mode limited to provisioning.	Must have; Full; V1-Y2.	Affects (vendor-supplied).	Not validated in the UC.
REQ38	KR8	CSE to have a "real" mode where resources are really provided and can be used for configuration and other further steps.	Must have; Full; V1-Y1.	Affects (vendor-supplied).	Not validated in the UC.
REQ39	KR8	CSE to enable extensibility (documented way): adding new mocked services, adding new "real" deployments.	Should have; Full; V1-Y2.	Affects.	Not validated in the UC.
REQ40	KR2	The IDE should provide a visual diagram functionality to visualise the different assets defined through the DOML and DOML Extensions.	Must have; Full; V1-Y1.	Affects.	Must have; Full; V1-Y1.
REQ41	KR2	The IDE should be extensible through plugin mechanism. Not only to support PIACERE assets (ICG, VT) but also for third party collaborators.	Must have; Full; V1-Y2.	Affects (vendor-supplied or ad-hoc solution).	Must have; Full; V1-Y2.
REQ42	KR2	The IDE should be implemented using open-source software.	Should have; Full; V1-Y1.	Affects (Desirable - vendor-supplied).	Could have; Full; V1-Y1.
REQ43	KR2	The IDE should be easily updatable to newer software versions.	Must have; Full; V1-Y1.	Affects.	Must have; Full; V1-Y1.
REQ44	KR2	The IDE could provide an import mechanism to automatically fulfil partial DOML.	Could have; Full; V1-Y2.	Affects (Desirable for efficiency).	Could have; Full; V1-Y1.
REQ46	KR9, KR11	The monitoring component shall gather metrics from the instances of the infrastructural elements at run time. These metrics need to be related to the TR and accessible to the IOP (through the dynamic part of the infrastructural catalogue).	Must have; Full; V1-Y1.	Affects.	Not validated in the UC.
REQ47	KR11	The monitoring component shall include the needed elements in the stack to monitor the infrastructural elements.	Must have; Full; V1-Y1.	Affects (Necessary).	Not validated in the UC.
REQ48	KR11	The monitoring component shall transform the real time values into the correct format/type/nature for the self-learning component.	Must have; Full; V1-Y2.	Affects (Necessary).	Not validated in the UC.
REQ50	KR11, KR12	The monitoring component shall monitor the metrics associated with the defined measurable TRs (e.g., performance, availability, and security through the runtime security monitoring).	Must have; Full; V1-Y1.	Affects.	Not validated in the UC.
REQ51	KR11, KR12	The self-learning component shall ensure that the conditions are met (compliance with respect to SLO) and that a failure or a non-compliance of a TRs is not likely to occur. This implies the compliance of a predefined set of non-functional requirements (e.g., performance).	Must have; Full; V1-Y1.	Affects (Desirable for performance, service availability, elasticity, other operational metrics).	Not validated in the UC.

REQ ID	KRs	Description	UC1	UC2	UC3
REQ52	KR11	Self-learning shall consume the data monitored and store it in a time-series database to create discriminative complex statistical variables and train a predictor which will learn potential failure patterns in order to prevent the system from falling into a TR violation situation.	Must have; Full; V1-Y1.	Affects.	Not validated in the UC.
REQ55	KR10	The IEM will log the whole IaC execution run, making metadata and metrics (time it took to run) about the creation of resources available to the rest of the PIACERE components.	Must have; Full; V1-Y2.	Desirable.	Must have; Full; V2-Y3.
REQ57	KR1	It is desirable to enable both forward and backward translations from DOML to IaC and vice versa.	Should have; Full; V1-Y1.	Desirable.	Should have; Full; V2-Y3.
REQ58	KR1	DOML should offer the modelling abstractions to define the outcomes of the IoP.	Must have; Full; V1-Y1.	Affects (Required).	Not validated in the UC.
REQ59	KR1	The DOML should allow users to define rules and constraints for redeployment, reconfiguration and other mitigation actions	Must have; Full; V1-Y2	Affects (Required).	Must have; Full; V1-Y2.
REQ60	KR1	DOML should support the modelling of security metrics both at the level of infrastructure and application.	Must have; Full; V1-Y1.	May affect (Desirable for full application -level integration).	Must have; Full; V1-Y2.
REQ61	KR1	DOML must support the modelling of NFRs and of SLOs.	Must have; Full; V1-Y1.	Affects.	Must have; Full; V1-Y2.
REQ62	KR1, KR2	DOML must support different views.	Should have; Full; V1-Y1.	Affects (Abstraction levels).	Should have; Full; V1-Y2.
REQ63	KR1	DOML must be unambiguous.	Must have; Full; V1-Y1.	Affects (Required and enforced).	Must have; Full; V1-Y1.
REQ64	KR2	The IDE should provide a text-based representation of DOML to ease version control.	Should have; Full; V1-Y2.	Affects (Desirable).	Should have; Full; V1-Y2.
REQ65	KR6, KR7	IaC Security Inspector and Component Security Inspector should hide specificities and technicalities of the current solutions in an integrated IDE.	Must have; Full; V1-Y1.	Desirable (built-in account privilege-based security by a need-to-know principle).	Must have; Full; V1-Y1.
REQ66	KR6	IaC Code security inspector must provide an interface (CLI or REST API) to integrate with other tools or CI/CD workflows.	Must have; Full; V1-Y1.	Desirable.	Must have; Full; V1-Y1.
REQ67	KR7	IaC Component security inspector must provide an interface (CLI or REST API) to integrate with other tools or CI/CD workflows.	Must have; Full; V1-Y1.	Desirable.	Must have; Full; V1-Y1.
REQ70	KR1	The DOML should allow users to state correctness properties in a suitable sub-language (possibly Formal Logic).	Must have; Full; V1-Y1.	Affects (Vendor-supplied or ad-hoc solution).	Must have; Full; V1-Y1.
REQ72	KR11	The runtime monitoring component should provide an UI for the end users to see the monitored resources and the corresponding metrics/TRs in real time.	Should have; Full; V1-Y2.	Affects (Desirable - vendor-supplied).	Not validated in the UC.
REQ76	KR1, KR2	DOML should allow the user to model each of the four considered DevOps activities (Provisioning, Configuration, Deployment, Orchestration).	Should have; Full; V1-Y1.	Affects (required for DevSecOps).	Should have; Full; V1-Y1.
REQ77	KR3	ICG may generate IAC code for different supported/target tools according to the required DevOps activity (as listed in REQ76).	Should have; Full; V1-Y1	Affects (required for	Should have; Full; V1-Y1

REQ ID	KRs	Description	UC1	UC2	UC3
				DevSecOps).	
REQ81	KR10	IEM should be able to execute IaC generated by ICG for selected IaC languages (e.g., TOSCA/Ansible/Terraform).	Must have; Full; V1-Y1 (at least Ansible).	Desirable (Vendor-supplied or ad-hoc solution).	Must have; Full; V1-Y1.
REQ82	KR10	IEM shall register the status of past and present executions and enable an appropriate way to query it.	Must have; Full; V1-Y2.	Desirable.	Must have; Full; V1-Y2.
REQ83	KR10	IEM should be able to communicate with the relevant actors (orchestrators, infrastructural elements) in a secure way.	Must have; Full; V1-Y2.	Desirable (DevSecOps, ENS).	Must have; Full; V1-Y2.
REQ84	KR10	IEM should be able to utilize the required credentials in a secure way.	Must have; Full; V2-Y3.	Desirable.	Must have; Full; V2-Y3.
REQ85	KR10	IEM should be able to clean up the resources being allocated.	Must have; Full; V1-Y2.	Desirable (required for efficiency-related garbage-collection).	Must have; Full; V1-Y2.
REQ87	KR10	IEM shall work against the production environment and the canary environment.	Must have; Full; V1-Y1.	Desirable.	Desirable; Full: V1-Y2.
REQ88	KR13	PIACERE framework should be usable by a team of people collaborating in the development of the same IaC.	Must have; Full; V1-Y2.	Affects (required).	Must have; Full V1-Y2.
REQ92	KR11	Self-healing component shall receive notifications from the self-learning.	Must have; Full; V1-Y1.	Affects (Required - useless otherwise).	Not validated in the UC.
REQ93	KR11	Self-healing component shall classify the events received from the self-learning and derive corrective actions.	Must have; Full; V1-Y1.	Affects (Required - useless otherwise).	Not validated in the UC.
REQ94	KR11	SelfHealing component shall inform the run time controller about the different components to orchestrate (the workflow to be executed).	Must have; Full; V1-Y2.	Affects (Required - useless otherwise).	Not validated in the UC.
REQ95	KR5	VT tools (model checker) must be able read DOML language.	Must have; Full; V1-Y1.	Affects.	Must have; Full; V1-Y2.
REQ96	KR3	ICG must be able read DOML language.	Must have; Full; V1-Y1.	Affects.	Must have; Full; V1-Y2.
REQ97	KR11	The SelfHealing components provide feedback on the DOML code, without doing automatic writes. The end user can choose to accept or not the feedback received.	Must have; Full; V1-Y1.	Affects.	Not validated in the UC.
REQ98	KR9	The IOP components provide feedback on the DOML code, without doing automatic writes. The end user can choose to accept or not the feedback received.	Must have; Full; V1-Y1.	Desirable.	Not validated in the UC.
REQ99	KR2	IDE to integrate with both local and remote Git repositories.	Must have; Full; V1-Y1.	Desirable.	Must have; Full; V2-Y3.
REQ100	KR3	ICG should generate IaC code that supports different cloud platforms.	Must have; Full; V1-Y1.	Affects (vendor-supplied)	Must have; Full; V1-Y1.
REQ101	KR3	IDE should allow to create and edit a graphical DOML model. Possibly starting from a palette of supported components that can be drag&drop in the graphical model.	Desirable; Full; V2-Y2.	Desirable; Full; V2-Y2.	Must have; Full; V1-Y2.
REQ103	KR5	Verification Tool (model checker) must verify the structural consistency of the DOML models.	Desirable; Full; V2-Y2.	Desirable; Full; V2-Y2.	Desirable; Full; V2-Y2.

REQ ID	KRs	Description	UC1	UC2	UC3
REQ104	KR5	Verification Tool (model checker) must verify the correctness of DOML models, with respect to some correctness properties provided in DOML.	Desirable; Full; V2-Y2	Desirable; Full; V2-Y2	Desirable; Full; V2-Y2
REQ105	KR5	Verification Tool (model checker) must verify the completeness of DOML models.	Desirable; Full; V2-Y2	Desirable; Full; V2-Y2	Desirable; Full; V2-Y2
REQ106	KR6, KR7	Organization of scan results with respect to the scan outcome. Aggregate the results of distinct scan tools in a form of unified results summary that makes distinction of the result into the following cases: passed – scan is performed without any issues detected.	Not validated in the UC.	Not validated in the UC.	Desirable; V2-Y3
REQ107	KR6, KR7	Improvement of scan response time. Compatibility matrix is adopted containing the list of available scans for various file types detected within IaC archive. Therefore, the execution of non-compatible scans for submitted IaC archive should be avoided.	Not validated in the UC.	Not validated in the UC.	Desirable; V2-Y3
REQ108	KR6, KR7	Management and persistence of scan results after the scan process is finished. Scan results are persisted into database for limited period (14 days), so users can look on their previous scan tasks and the achieved outcomes.	Not validated in the UC.	Not validated in the UC.	Desirable; V2-Y3
REQ109	KR6, KR7	Scan configuration management. There should be a possibility to maintain multiple scans concurrently for multiple users.	Not validated in the UC.	Not validated in the UC.	Desirable; V2-Y3
REQ110	KR3	ICG should provide enough extensibility to: comply with the DOML extension mechanism; be capable of integrating new IaC languages.	Not validated in the UC.	Not validated in the UC.	Not validated in the UC.

2.4 Requirements Summary Dashboard

The following Table 7 summarizes how the requirements are distributed among KRs and work packages.

This table shows the association between KRs and WPs and the planning of the implementation of the requirements during the 3 years of the PIACERE project.

The major development efforts were made in Y1, to allow the first integration tests done in Y2. This explains the higher number of requirements met in Y1. In Y2 and Y3 we are focusing more on the Integration between KRs and on UCs validation and finally on the residual part of the requirements to be achieved.

Table 7: PIACERE Requirements Summary Table

KR	WP	Accepted requirements	Y1	Y2	Y3
KR1	WP3	17	12	5	0
KR2	WP3	11	7	4	0
KR3	WP3	5	3	1	1
KR4	WP3	5	3	2	0
KR5	WP4	4	4	0	0
KR6	WP4	7	3	0	4
KR7	WP4	8	4	0	4
KR8	WP5	5	3	2	0
KR9	WP5	4	2	2	0
KR10	WP5	8	2	6	0
KR11	WP6	14	9	5	0
KR12	WP6	9	5	4	0
KR13	WP2	2	0	2	0

In Figure 4 it is shown in a more intuitive way the accepted requirements distribution among KR.s.

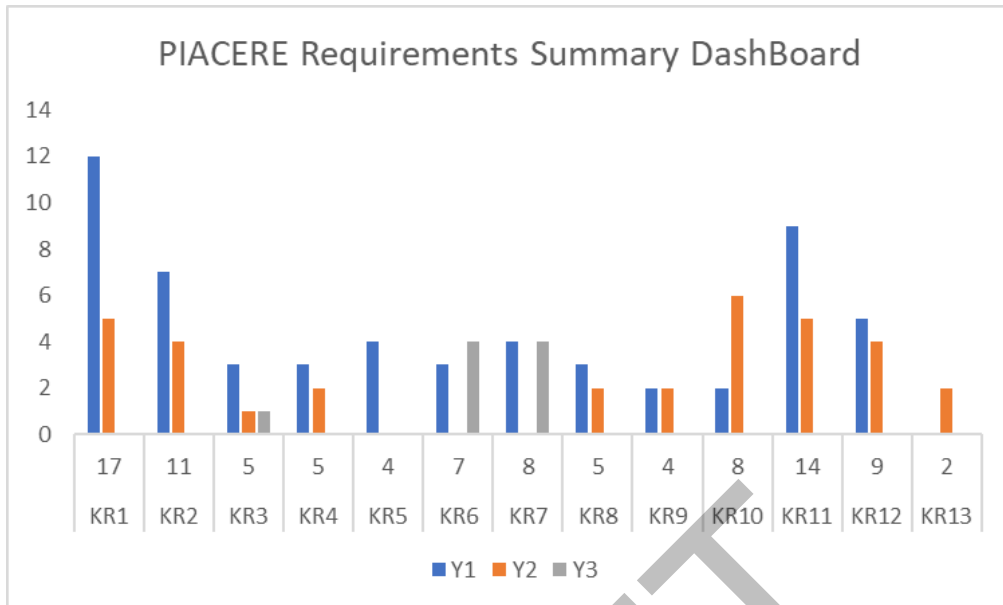


Figure 4: PIACERE Requirements Summary Dashboard

DRAFT

3 PIACERE Architecture

The PIACERE Architecture, whose purpose is to support the modelling and creation of the infrastructure an application is running upon, is structured in blocks that correspond to the PIACERE Key results from KR1 to KR12 (see Figure 1), composing the final KR13, that is the PIACERE DevSecOps Framework.

3.1 Changes in v2

This section reports the updates in the outcome of architecture definition task in the second year of the project.

The general architecture workflows of design and runtime phases have been improved since the submission of deliverable D2.1 and updated in the communication mechanisms between some KRs, identifying requests, responses and user interactions (Figure 5, Figure 7 and Figure 8). The description of each interaction in Table 8, Table 9 and Table 10 has been presented more effectively specifying the activity. These updates have resulted in the complete revision of section 3.3.

In section 3.4 the architecture of some components and the presentation of some sequence diagrams describing the functioning of the KRs, has been updated.

Some components can be involved in both design and runtime phases: ICG (KR3) is mainly involved in the design phase, but it can also be invoked in the runtime phase; similarly, IOP (KR9) is mainly involved in the runtime phase, but it can be invoked in the design phase. The sequence diagrams of ICG, IOP, IDE have been changed accordingly in sections 3.4.3, 3.4.8.1 and 3.4.1 respectively.

The IaC Scan Runner component has been added in section 3.4.4 to manage the security checks related to IaC Security Inspector (KR6) and Component security inspector (KR7).

Self-healing mechanism and sequence diagram (Figure 27) have been updated: if the self-learning or the monitoring component detects a failure or potential failure, the self-healing component receives an alerting message. The self-healing component categorizes the incidence to start the appropriate workflows (redeploy, scale, quarantine) calling the PRC. The other sequence diagrams of the monitoring components have been updated accordingly.

PRC remains the sole contact between the design time (IDE) and runtime tools: the sequence diagrams of IEM, IDE have been changed accordingly in sections 3.4.5 and 3.4.1 respectively.

The sequence diagram of Infrastructural Elements Catalogue (Figure 28) has been updated to point out its role in design time and runtime phases.

Overall, the section 3.4 reports about updates to the PIACERE components based on the development in the second year of the project

An overview to the multi-user approach has been presented in section 3.5. Some scenarios of the using of PIACERE framework from the user perspective have been described in section 3.7.

3.2 General description

The PIACERE DevSecOps framework (KR13) is the integration point for all PIACERE Key Results. It provides three main functionalities:

1. It serves as entry point to PIACERE. A user wishing to utilize the tools will do so through the DevSecOps framework.

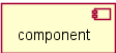
2. It integrates the different tools and KRs.
3. It orchestrates the workflow, supporting the integrated continuous development and operation approach. The DevSecOps framework will launch the appropriate tool for each phase of the application's lifecycle.

The main entry point of the framework is the GUI provided by IDE (KR2) that drives the design phase and acts as a gateway to the runtime phase.

3.3 Logical/Functional View

The PIACERE architecture can be divided into two macro areas called "*Design*" and "*Runtime*".

In Figure 5, Figure 7 and Figure 8 the interaction among the components of the PIACERE framework in a typical workflow is shown for both areas.

In these figures, we have the components represented by  symbol and two different kinds of flows:

- *Request*, represented by the solid line, to indicate a call from a component to the next one.
- *Response*, represented by the dashed line, to indicate the response sent back to the component needed for the next step.

The PIACERE *Design* architectural area describes the components that carry out the design and planning phase of the automation code providing the user with the tools to design, plan, create, verify the trustworthiness of IaC as well packing it for the deployment.

The PIACERE *Design* time, as shown in Figure 5 is composed by the following components:

- Integrated Development Environment (IDE, KR2)
- Verification Tool (VT) which includes Model Checker (KR5) and the two components IaC Security Inspector (KR6), Component Security Inspector (KR7) grouped in IaC Scan Runner that acts as KR6-KR7 executor
- Infrastructural Code Generator (ICG, KR3)
- IaC Optimizer Platform (IOP, KR9)

PIACERE uses a proprietary modelling language, called DOML (KR1), represented in the Figure 5 by the green box. This language includes the DOML extension mechanisms, called DOML-E (KR4), concerning the ability of PIACERE users to extend the DOML elements and to support to new IaC languages.

ICG is mainly involved in the design phase, but it can also be invoked in the runtime phase. Similarly, IOP is mainly involved in the runtime phase, but it can be invoked in the design phase.

PIACERE Data Repository consists of:

- "DOML and IaC repository"
- "Infrastructural Elements Catalogue"

"DOML and IaC repository" stores DOML models and IaC code while the "Infrastructural Elements Catalogue" is a repository for storing the description of the infrastructure elements together with their historical and statistical data.

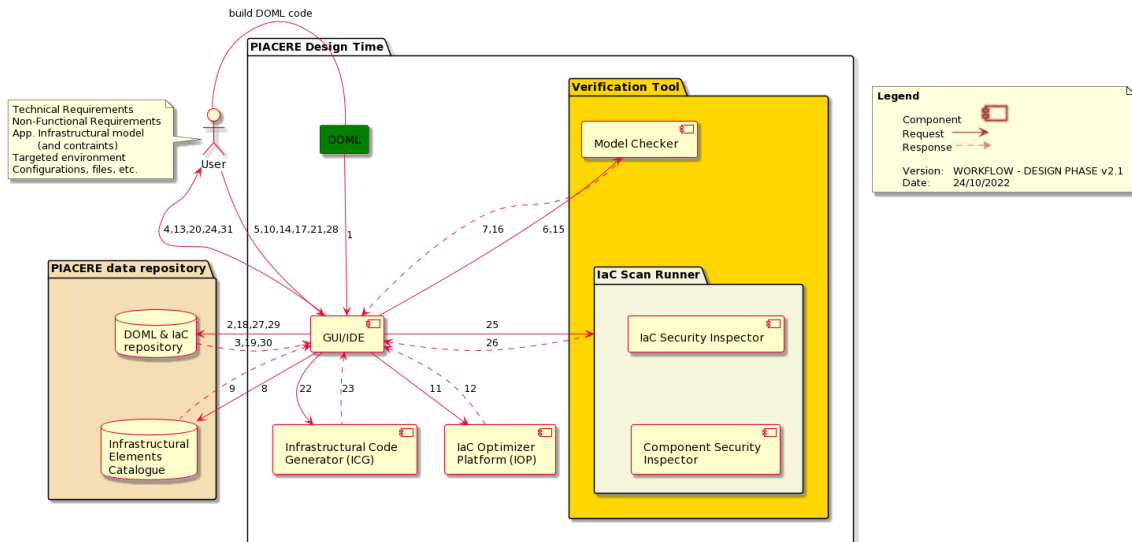


Figure 5: PIACERE Design Time

Table 8 describes the design steps allowing a user to create and save new DOML model(s) and correlated infrastructural elements in the PIACERE Data Repository.

The **Arrow #** column corresponds to the arrows of Figure 5 and also to the step, the **From** and **To** are respectively the starting and ending point of the flow, **Interaction** could be *Request* to indicate a call to a PIACERE component with any data needed, *Response* to indicate a response to a previous request or *User Interaction* when the user interaction is expected, finally **Description** describes the step.

Table 8: PIACERE Design Workflow

Arrow #	From	To	Interaction	Description
1	User	GUI IDE	User interaction	DOML generation: User interacts with GUI IDE and can start design DOML (build or import DOML).
2	GUI IDE	DOML&IaC Repository	Request	DOML generation: IDE search DOML model in the repository.
3	DOML&IaC Repository	GUI IDE	Response	DOML generation: IDE gets DOML information from the repository.
4	GUI IDE	User	User Interaction	DOML generation: The user inspects DOML model.
5	User	GUI IDE	User Interaction	DOML verification: the user requests DOML verification.
6	GUI IDE	Model Checker	Request	DOML verification: Verify DOML model.
7	Model Checker	GUI IDE	Response	DOML verification: Model Checker return an answer to IDE. In case of positive answer go to next step otherwise the process restart.
8	GUI IDE	Infrastructural elements catalogue	Request	Targeted environment information to be considered in the optimization process by the IOP.
9	Infrastructural elements catalogue	GUI IDE	Response	Targeted environment information acquired.
10	User	GUI IDE	User Interaction	DOML optimization (optional): The user requests optimization of DOML.
11	GUI IDE	IOP	Request	DOML optimization (optional): IDE requests to IOP to optimize DOML model.
12	IOP	GUI IDE	Response	DOML optimization (optional): IOP send back the optimized DOML.

Arrow #	From	To	Interaction	Description
13	GUI IDE	User	User Interaction	DOML optimization (optional): the user can inspect the optimized DOML.
14	User	GUI IDE	User Interaction	DOML optimization (optional): DOML verification request.
15	GUI IDE	Model Checker	Request	DOML optimization (optional): Verify DOML model.
16	Model Checker	GUI IDE	Response	DOML optimization (optional): Model Checker return an answer to IDE.
17	User	GUI IDE	User Interaction	The user commits the changes.
18	GUI IDE	DOML & IaC Repository	Request	IDE saves the new version in the repository.
19	DOML & IaC Repository	GUI IDE	Response	IDE receives feedback (new version pushed).
20	GUI IDE	User	User Interaction	IDE sends the acknowledge to the user.
21	User	GUI IDE	User Interaction	IaC Generation: Request to generate IaC
22	GUI IDE	Infrastructural Code Generator (ICG)	Request	IaC Generation: IDE calls ICG with the request to generate IaC. ICG may generate IaC for different tools/languages, according to the DevOps activity to be automated.
23	Infrastructural Code Generator (ICG)	GUI IDE	Response	IaC Generation: ICG generates code based on the received DOML.
24	GUI IDE	User	User Interaction	IaC Generation: IDE gives feedback on the IaC code generated.
25	GUI IDE	IaC Scan Runner	Request	IaC Security Inspection: IaC Security Inspector and Component Security Inspector checks the code, the cryptographic libraries and the configuration files provided.
26	IaC Scan Runner	GUI IDE	Response	IaC Security Inspection: IaC Security Inspector and Component Security Inspector return a set of warnings, errors and recommendations to the GUI.
27	GUI IDE	DOML & IaC Repository	Request	IaC code is saved into IaC Repository.
28	User	GUI IDE	User Interaction	Add/commit repository changes.
29	GUI IDE	DOML & IaC Repository	Request	Commit the new version in the DOML & IaC Repository.
30	DOML & IaC Repository	GUI IDE	Response	New version committed.
31	GUI IDE	User	User interaction	Push notification to the user.

Figure 6 shows the sequence of use of the components involved in the design phase. The colours chosen to identify the components indicate the type of action they perform (as shown in the legend in the picture).

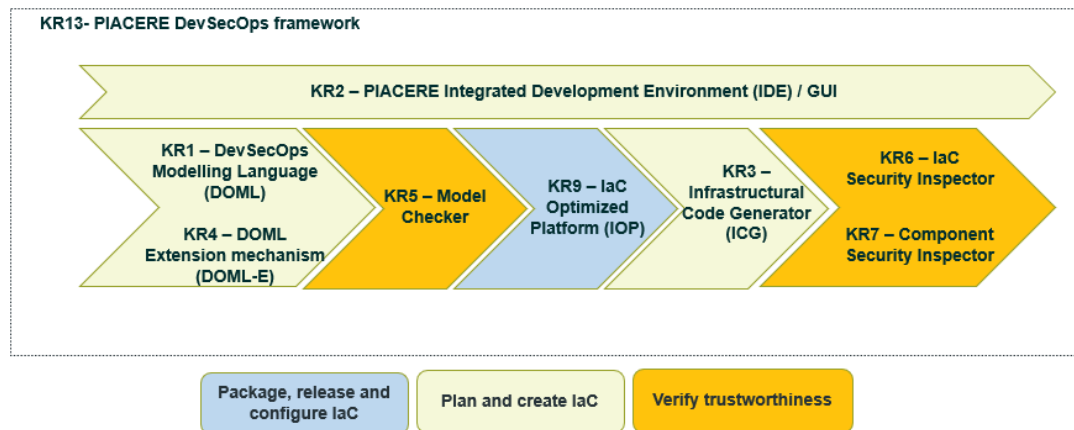


Figure 6: KRs involved in PIACERE Design Time

The PIACERE *Runtime* architectural area describes the components necessary for automated deployment, for the dynamic environment that is created during the deployment phase itself and for the infrastructure resource monitoring activation and deactivation activities.

To simplify the reading of the PIACERE *Runtime* workflow, the PIACERE *Runtime* diagrams have been divided according to the different component roles:

- Release, configure, check, and deploy (Figure 7)
- Monitor, self-healing, self-learning (Figure 8)

The PIACERE Runtime, as shown in Figure 7 and Figure 8, is composed by the following components:

- Runtime Controller (PRC)
- IaC Executor Manager (IEM, KR10)
- Resource Provider
- Canary Sandbox Environment Provisioner
- Canary Sandbox Environment Mocklord
- Infrastructure Advisor
 - IDE Plug-in/Dashboard
 - IaC Optimizer Platform (IOP, KR9)
 - Monitoring Controller
 - Performance Monitoring (KR12)
 - Security Monitoring (KR12)
 - Performance Self-Learning (KR11)
 - Security Self-Learning (KR11)
 - Self-Healing (KR11)

The IDE and the PIACERE data repository have been already described above for the Design phase, IDE also supports users during the Runtime phase. The PIACERE PRC acts as the runtime workflow engine.

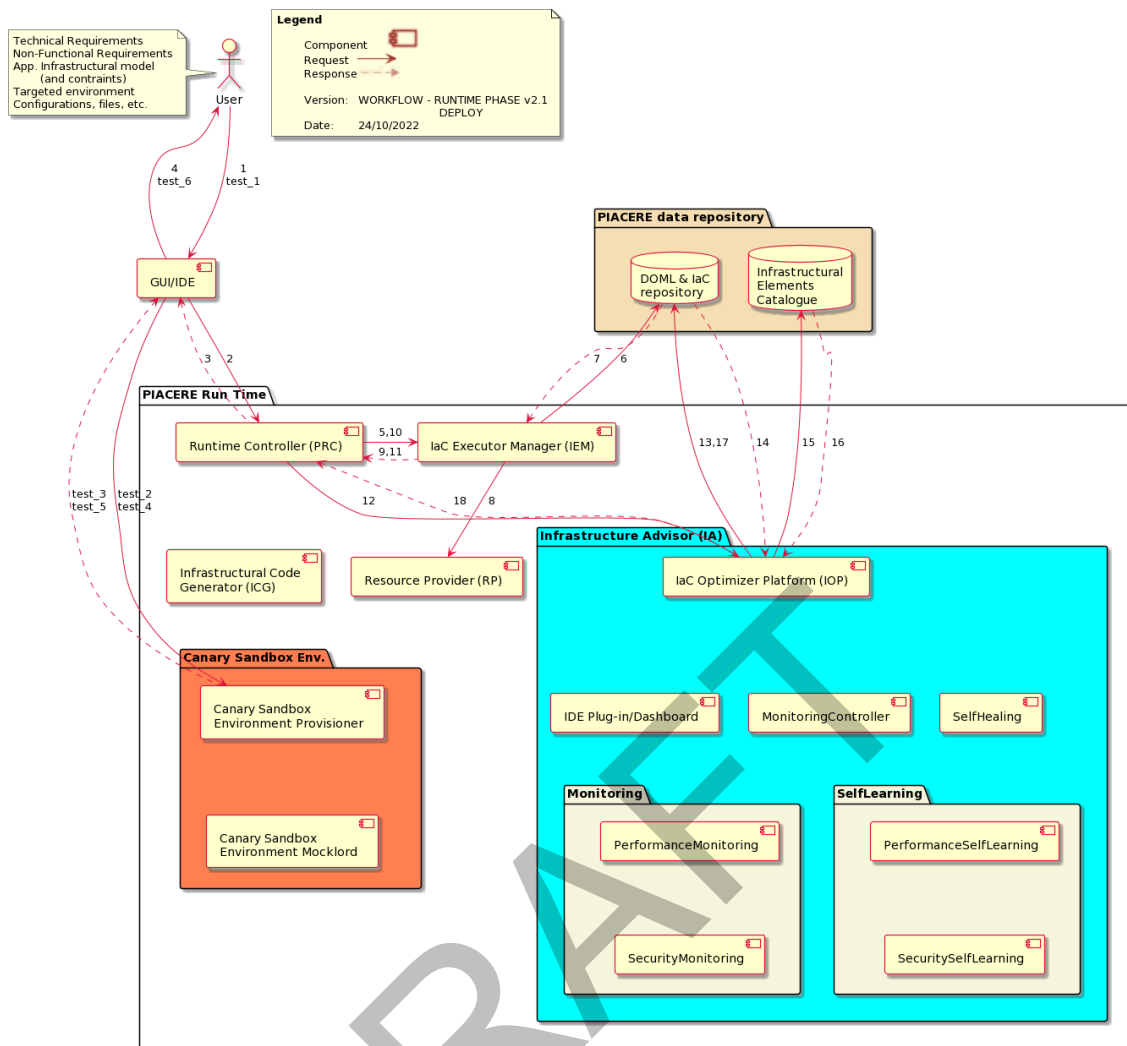


Figure 7: PIACERE Runtime – Deployment activities

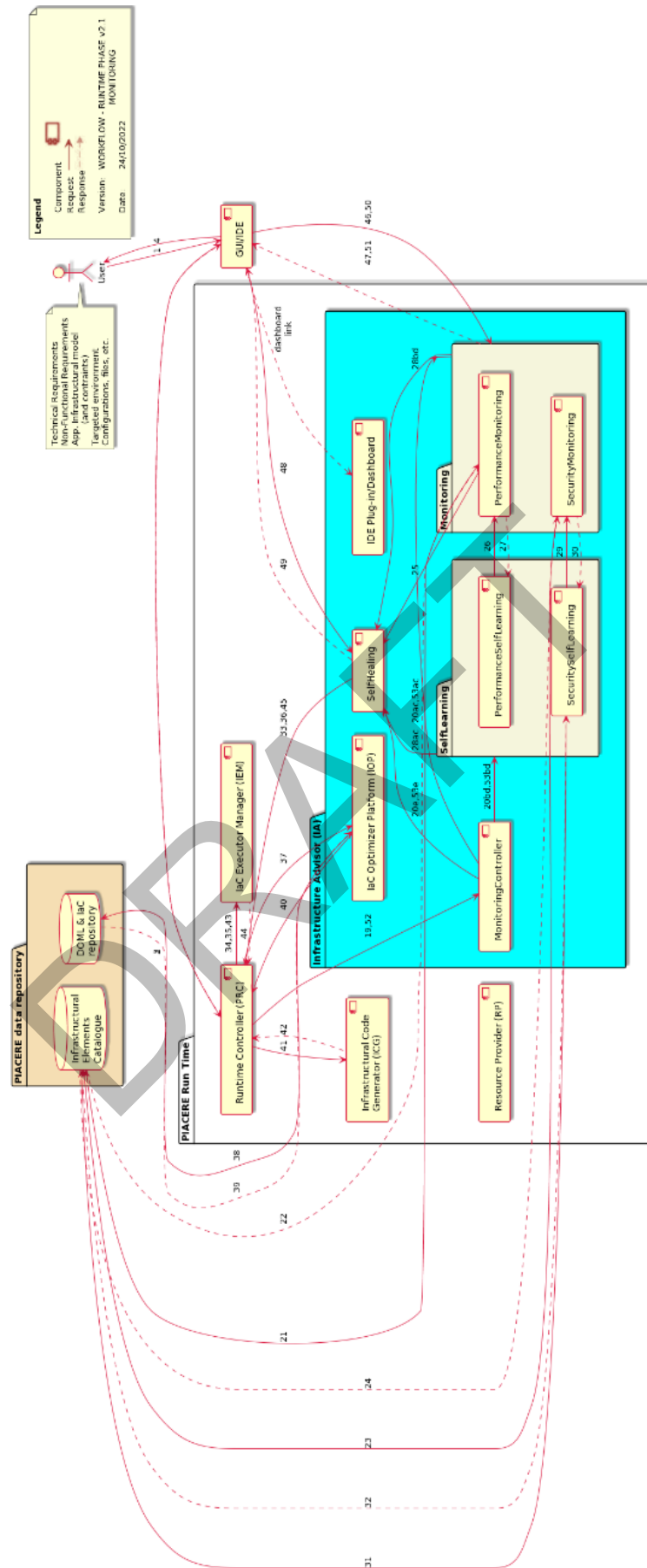


Figure 8: PIACERE Runtime – Monitoring activities

Table 9 describes the runtime steps allowing a user to implement and manage the execution and the monitoring of the environment; Table 10 describes the runtime steps allowing a user to dynamically test the IaC using the Canary environment.

The **Arrow #** column corresponds to the arrows of Figure 7, Figure 8 and also to the step of the process, the **From** and **To** are respectively the starting and ending point of the flow, **Interaction** could be *Request* to indicate a call to a PIACERE component with any data needed, *Response* to indicate a response to a previous request or *User Interaction* when the user interaction is expected, finally **Description** describes the step of the process.

There are cases where arrow numbers are identical with an additional label (e.g. 20a, 20b), this means simultaneous actions and steps. The arrow numbers preceded by “test” in Table 10 refer to interactions related to the Canary environment.

Table 9: PIACERE Runtime Workflow

Arrow #	From	To	Interaction	Description
1	User	GUI IDE	User interaction	Interaction User-IDE: User interacts with Infrastructure Advisor and vice versa via GUI IDE.
2	GUI IDE	Runtime Controller (PRC)	Request	Interaction User-IDE: IDE activates the PRC with the request from the user.
3	Runtime Controller (PRC)	GUI IDE	Response	Interaction User-IDE: At the end of the requested activity, the PRC sends the response to the IDE.
4	GUI IDE	User	User Interaction	Interaction User-IDE: User receives the response via GUI IDE.
5	Runtime Controller (PRC)	IaC Executor manager (IEM)	Request	Deployment activities: Deployment request.
6	IaC Executor manager (IEM)	DOML & IaC Repository	Request	Deployment activities: IaC Deployment request.
7	DOML & IaC Repository	IaC Executor manager (IEM)	Response	Deployment activities: IaC Deployment response.
8	IaC Executor manager (IEM)	Resource Provider (RP)	Request	Deployment activities: Deployment commands.
9	IaC Executor manager (IEM)	Runtime Controller (PRC)	Response	Deployment activities: Deployment response.
10	Runtime Controller (PRC)	IaC Executor manager (IEM)	Request	Deployment status activities: Deployment status request.
11	IaC Executor manager (IEM)	Runtime Controller (PRC)	Response	Deployment status activities: Deployment status response.
12	Runtime Controller (PRC)	IaC Optimizer Platform (IOP)	Request	Optimization process: Launch new optimization process.
13	IaC Optimizer Platform (IOP)	DOML & IaC Repository	Request	Optimization process: IOP requests information about optimization requirements and objectives.
14	DOML & IaC Repository	IaC Optimizer Platform (IOP)	Response	Optimization process: IOP receives information about optimization requirements and objectives.

Arrow #	From	To	Interaction	Description
15	laC Optimizer Platform (IOP)	Infrastructure l elements catalogue	Request	Optimization process: IOP requests targeted environment information.
16	Infrastructural elements catalogue	laC Optimizer Platform (IOP)	Response	Optimization process: IOP receives targeted environment information.
17	laC Optimizer Platform (IOP)	DOML & laC Repository	Request	Optimization process: provides feedback about the new deployment configuration.
18	laC Optimizer Platform (IOP)	Runtime Controller (PRC)	Response	Optimization process: optimization finished.
19	Runtime Controller (PRC)	Monitoring Controller	Request	Monitoring process: start monitoring for a given application by means of PRC.
20a	Monitoring Controller	Performance Monitoring	Request	Monitoring process: start monitoring for a given application.
20b	Monitoring Controller	Performance SelfLearning	Request	Monitoring process: start monitoring for a given application.
20c	Monitoring Controller	Security Monitoring	Request	Monitoring process: start monitoring for a given application.
20d	Monitoring Controller	Security SelfLearning	Request	Monitoring process: start monitoring for a given application.
20e	Monitoring Controller	SelfHealing	Request	Monitoring process: start monitoring for a given application.
21	Performance Monitoring	Infrastructure l elements catalogue	Request	Monitoring process: request of performance information related to the infrastructure elements to support IOP algorithms.
22	Infrastructural elements catalogue	Performance Monitoring	Response	Monitoring process: return of requested data.
23	Security Monitoring	Infrastructure l elements catalogue	Request	Monitoring process: request of security information related to the infrastructure elements to support IOP algorithms.
24	Infrastructural elements catalogue	Security Monitoring	Response	Monitoring process: return of requested data.
25	Performance Monitoring	SelfHealing	Request	Monitoring process: send a “notify event” in cases a warning threshold has been raised by monitoring.
26	Performance SelfLearning	Performance Monitoring	Request	Monitoring process: request of timeseries data.
27	Performance Monitoring	Performance SelfLearning	Response	Monitoring process: timeseries of different performance metric provided (eh.g. memory, disk usage, etc).
28a	Performance SelfLearning	SelfHealing	Request	Monitoring process: send a “notify event” in cases a warning threshold has been raised by monitoring.
28b	Performance Monitoring	SelfHealing	Request	Monitoring process: send a “notify event” in cases a warning threshold has been raised by monitoring.
28c	Security Monitoring	SelfHealing	Request	Monitoring process: send a “notify event” in cases a warning threshold has been raised by monitoring.
28d	Security SelfLearning	SelfHealing	Request	Monitoring process: send a “notify event” in cases a warning threshold has been raised by monitoring.
29	Security SelfLearning	Security Monitoring	Request	Security monitoring process: acquire data.
30	Security Monitoring	Security SelfLearning	Response	Security monitoring process: data continuous.

Arrow #	From	To	Interaction	Description
31	Security SelfLearning	Infrastructura l elements catalogue	Request	Security monitoring process: model training (optional) - store model to storage.
32	Infrastructural elements catalogue	Security SelfLearning	Response	Security monitoring process: model training (optional) - store data acknowledge.
33	SelfHealing	Runtime Controller (PRC)	Request	Self-healing process: propose to redeploy the workflow.
34	Runtime Controller (PRC)	laC Executor manager (IEM)	Request	Self-healing process: destroy deployment.
35	Runtime Controller (PRC)	laC Executor manager (IEM)	Request	Self-healing process: execute laC (restart deployment).
36	SelfHealing	Runtime Controller (PRC)	Request	Self-healing process: propose to scale the workflow.
37	Runtime Controller (PRC)	laC Optimizer Platform (IOP)	Request	Self-healing process: launch new optimization process.
38	laC Optimizer Platform (IOP)	DOML & laC Repository	Request	Self-healing process: store new deployment configuration.
39	DOML & laC Repository	laC Optimizer Platform (IOP)	Response	Self-healing process: acknowledge previous request.
40	laC Optimizer Platform (IOP)	Runtime Controller (PRC)	Request	Self-healing process: optimization finished.
41	Runtime Controller (PRC)	Infrastructura l Code Generator (ICG)	Request	Self-healing process: create new laC.
42	Infrastructural Code Generator (ICG)	Runtime Controller (PRC)	Response	Self-healing process: done.
43	Runtime Controller (PRC)	laC Executor manager (IEM)	Request	Self-healing process: execute new laC.
44	laC Executor manager (IEM)	Runtime Controller (PRC)	Response	Self-healing process: new laC executed.
45	SelfHealing	Runtime Controller (PRC)	Request	Self-healing process: Propose user feedback workflow.
46	GUI IDE	Performance Monitoring	Request	Monitoring process: request Dashboard.
47	Performance Monitoring	GUI IDE	Response	Monitoring process: IDE received the dashboard link.
48	GUI IDE	SelfHealing	Request	Self-healing process: request Dashboard.
49	SelfHealing	GUI IDE	Response	Self-healing process: IDE received the dashboard link.
50	GUI IDE	Security Monitoring	Request	Security monitoring process: Request Dashboard.
51	Security Monitoring	GUI IDE	Response	Security monitoring process: IDE received the dashboard link.
52	Runtime Controller (PRC)	Monitoring Controller	Request	Monitoring process: request to stop monitoring for a given application.

Arrow #	From	To	Interaction	Description
53a	Monitoring Controller	Performance Monitoring	Request	Monitoring process: request to stop monitoring for a given application.
53b	Monitoring Controller	Performance SelfLearning	Request	Monitoring process: request to stop monitoring for a given application.
53c	Monitoring Controller	Security Monitoring	Request	Monitoring process: request to stop monitoring for a given application.
53d	Monitoring Controller	Security SelfLearning	Request	Monitoring process: request to stop monitoring for a given application.
53e	Monitoring Controller	SelfHealing	Request	Monitoring process: request to stop monitoring for a given application.

Table 10: PIACERE Test Workflow

Arrow #	From	To	Interaction	Description
test 1	User	GUI IDE	User interaction	Interaction User-IDE: User interacts with Canary Sandbox Environment to request a new deployment via GUI IDE.
test 2	GUI IDE	Canary Sandbox Environment Provisioner	Request	Request new deployment
test 3	Canary Sandbox Environment Provisioner	GUI IDE	Response	Respond with new deployment identifier
test 4	GUI IDE	Canary Sandbox Environment Provisioner	Request	Register to watch for that new deployment entry status change
test 5	Canary Sandbox Environment Provisioner	GUI IDE	Response	Notify finished deployment
test 6	GUI IDE	User	User Interaction	Interaction User-IDE: User receives the response via GUI IDE.

In Figure 9 the main role of PIACERE KRs that compose the Runtime area is shown:

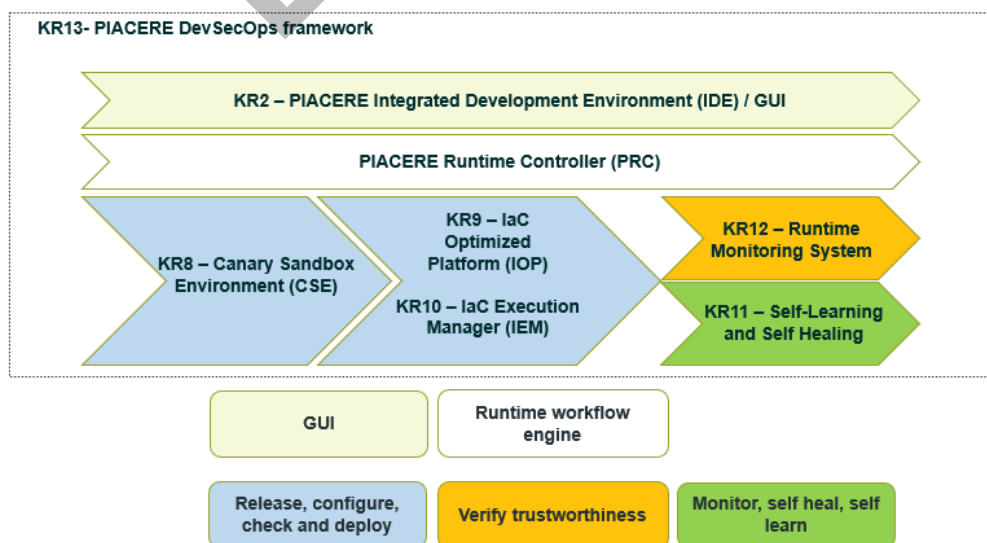


Figure 9: KRs involved in PIACERE Runtime

3.4 Architecture components

The purpose of this chapter is to describe in detail all the functional and non-functional components of the PIACERE architecture.

For each of the following sections, *Component description* has the aim to describe the component, its functions, any subdivisions of the same and everything necessary to correctly indicate it; *Component behavioural description* aims to describe the behaviour of the component with the other components, internal and external.

This section presents all KR updated with the changes performed in Y2. The most significant changes are on KR2, KR3, KR6, KR7 and KR9.

Overall, this section reports about updates to the PIACERE components according to the development in the second year of the project.

These developments have also led to a revision of the sequence diagrams, with the exception of those relating to the KR5, KR8 and KR10.

3.4.1 Integrated Development Environment - IDE (KR2)

Component Description

The PIACERE IDE (Integrated Development Environment) is a tool for modelling and verifying IaC solutions following the Model-Driven Engineering (MDE) approach. The IDE will enable to define IaC at an abstract level independently of the target environment and at concrete level, based on the PIACERE DOML (DevOps Modelling Language) and DOML-E (DOML Extensions).

At the technological level, the IDE has been developed using the Eclipse Modelling Framework, a technology developed to create own tools or IDEs and to describe metamodels. The IDE is the main tool for interaction with PIACERE users and acts as a vertebral element of the project. It has a user interface that allows interaction with other PIACERE tools/components. The IDE is set to be extensible by design, so to allow the new IaC tools and the new abstractions of infrastructural components that will be incorporated into DOML as Extensions.

Component behavioural description

The IDE, as the main interface for user's interaction, is connected with other PIACERE tools/components. The design time components are more tightly integrated with the IDE as they all belong to the design phase of the solution and make intensive use of the DOML. The other components belong to the runtime phase and are less coupled with the IDE, but nevertheless the IDE is still the summoning point for these components, and the communication between them is done through different communication interfaces such as REST APIs.

Through the IDE, users can describe their models according to the underlying metamodel, which in the case of PIACERE is the DOML. The model will contain the abstract and the concrete specification of the problem/project.

The IDE will integrate the Verification Tool (VT) and the Infrastructural Code Generator (ICG). Thanks to the VT, it will be possible to validate the defined models and to make suggestions, possible substitutions, and improvements. The ICG tool, when triggered from the IDE, will automatically obtain the corresponding IaC in a specific target environment (e.g., Terraform, Ansible, TOSCA, ...) from DOML.

All the information produced at design time will be stored into the PIACERE data repository, and after finalising the design time phase, a DOML specification will be complete, and an IaC of the

project will be generated. Before generating the IaC, it is possible to make a call to the IOP, to obtain which is the best solution for the concretization layer, setting a series of criteria in the call. This returns several solutions that the user can check and select the one he/she considers most appropriate and later, call the VT to ensure that everything is OK and, once this is done, call the ICG to obtain the IaC from DOML.

The runtime components of the PIACERE will be also linked with the IDE. The runtime controller (PRC) will be invoked through the IDE. This component will oversee doing the deployments and link them with the Infrastructure Advisor components.

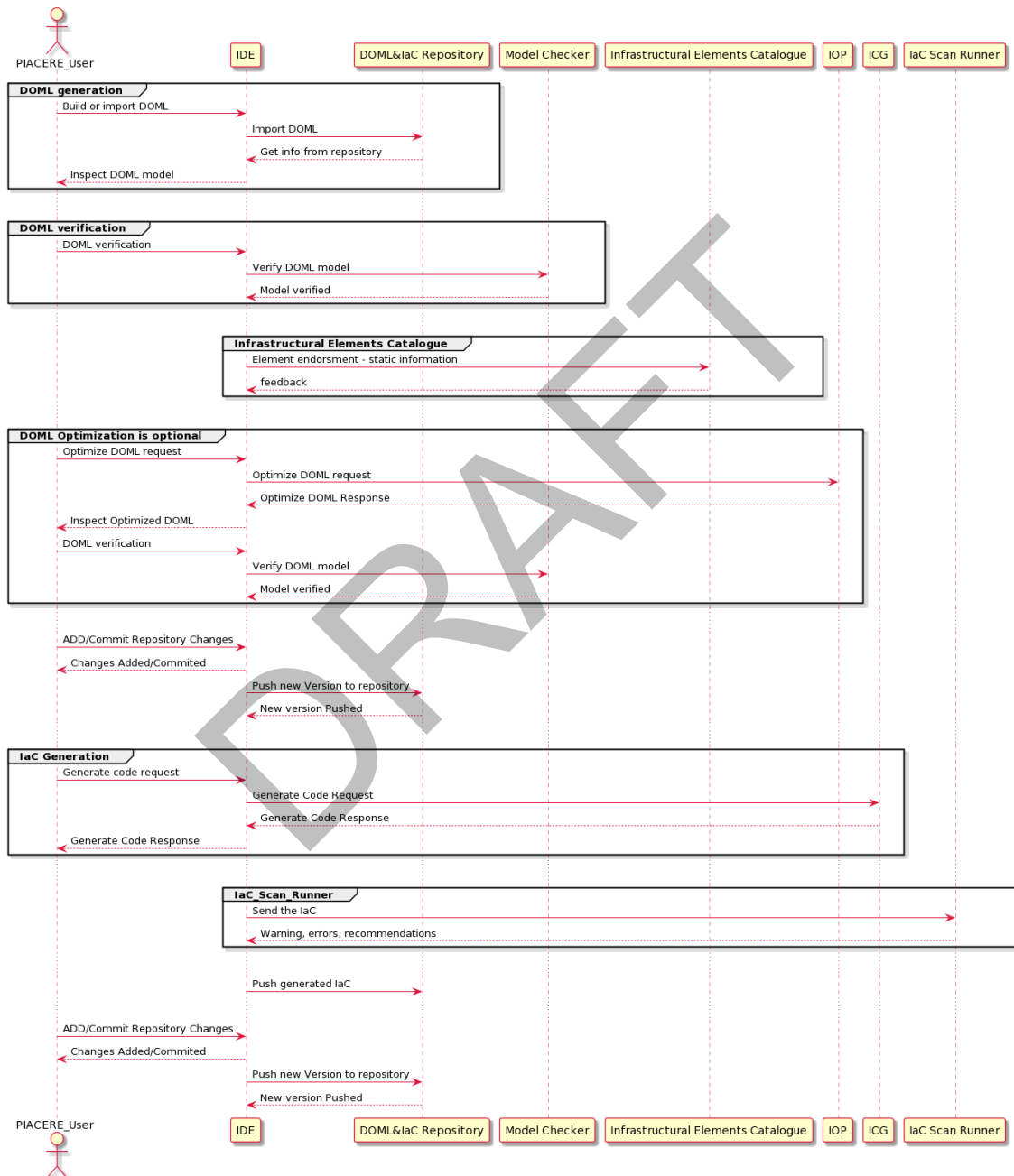


Figure 10: IDE sequence diagram

Figure 10 shows the interaction of the IDE with the different components of the design stage. Following the normal workflow, the first stage is the generation of the DOML, user can create a new file or import an existing one from a repository. Next step is the verification of the created

DOML. After this, the user can consult the catalogue of the elements to check the available elements for the deployment, but they can also define their optimization criteria and call the IOP that will return several options for concretization of the infrastructure and the user must select one and re-validate calling VT again. Once this is done you could save it in the repository to keep it and call the code generator that will return the corresponding IaC that can be scanned by the IaC Scan Runner to ensure the absence of errors.

3.4.2 DevOps Modelling Language – DOML/DOML-E (KR1-KR4)

Component Description

The DOML is the modelling language that is being defined to help PIACERE users in defining the deployment-relevant information concerning their software system.

The usage of the language is supported by a subcomponent in the IDE which includes those data structures representing the main elements that are part of the language. This subcomponent offers to the user a suggestion-based editing approach. More specifically, through the IDE, the user creates a DOML file and starts editing it. Based on what he/she is typing, the DOML component suggests how to complete the specification fragment and creates in memory the instances of the corresponding linguistic elements. These can be queried and serialized in a textual, XML or JSON file. The XML serialization (we call it DOMLx) has been implemented and is used as interchange format between the PIACERE tools.

The DOML extension mechanisms concern the ability of PIACERE users to extend the DOML in the following directions:

- **Creation of new DOML elements:** The types of computational nodes that can be adopted for hosting an application component, as well as the resources used to interconnect computational nodes and to control their execution can vary depending on the new technological advantages. To enable the PIACERE expert users to represent these new resources in the DOML, it should be possible to extend the language. Such extension should be similar to the type of creation mechanism offered by typical programming languages. In the second project year this mechanism has been used to accommodate the needs of the PIACERE case studies. The procedure to extend the DOML editor to support suggestion-based capabilities in this case has been manually followed. The possibility to let the user develop such extension by himself/herself is under study.
- **Extend current DOML elements:** This feature allows the user to add new attributes and properties to currently existing DOML elements. This aspect has been implemented and is currently being tested within the PIACERE case studies.
- **Support to new IaC languages:** The objective of this feature is to allow users to exploit new IaC languages for performing specific actions. We are studying two possibilities. The first one concerns the possibility to incorporate into the DOML references to code fragments/scripts written in specific IaC languages. The second one concerns the possibility to generate from a DOML specification scripts into multiple IaC languages. In the second project year we have addressed the first point by introducing in the language a new concept that allows the user to refer through a URI to specific code fragments and to explicitly indicate the executor to be used for running that code fragment. The second point has been addressed by the ICG that is currently able to generate both Terraform and Ansible code from a DOML specification. The possibility to generate code in other languages is being analysed and will be an aspect tackled in the last project year.

Component behavioural description

Figure 11 below shows the interaction between the user and the DOML, mediated by the IDE. More specifically, the figure highlights two logical subcomponents of DOML, the DOML Manager and the DOML Model. The first one is in charge of managing the interaction with the user, while the second one represents the set of data structures defining the DOML. When the user creates a DOML model through the IDE, it activates the DOML Manager which, in turn, instantiates a new DOML Model. The DOML Manager is the engine that from the knowledge of the DOML structure (the entities to be modelled and the needed relationships among them) ensures that a DOML Model is created properly. The DOML Manager includes editing features. Moreover, it helps the PIACERE user in his/her work by providing proper suggestions. Whenever the user adds a new DOML element, the corresponding object is created in the DOML Model. The interaction with the user can continue alternating suggestions, insertions of new DOML elements as well as modifications of existing elements. From time to time, the user will save the model, this operation will result in a serialization of the model into an XML, JSON or pure textual format. Finally, through the IDE, the user will push the model into a proper repository.

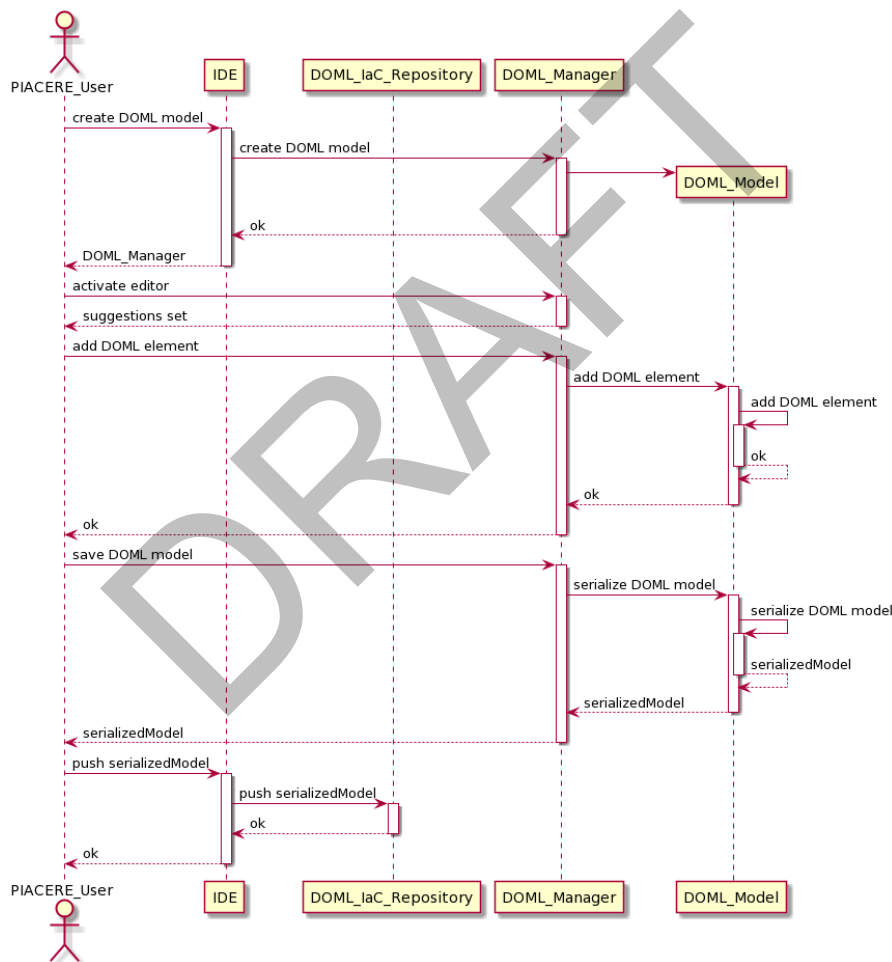


Figure 11: Interaction of the PIACERE User with DOML and the IDE

3.4.3 Infrastructural Code Generator - ICG (KR3)

Component Description

The Infrastructural Code Generator (ICG) is the PIACERE component that allows generating executable infrastructural code (IaC) from models written in DOML. ICG is a microservice application inside the PIACERE framework and is called through REST API: it takes the source

DOML model in the XML format (DOMLX) as input and produces IaC code files as output. The previous version of the ICG was developed as a command line interface compiler, as such the source code of the ICG retains the capability of being called through command line. In the current version, the ICG is extended and now it adopts REST API interfaces too and is aligned with the PIACERE microservices approach. This new solution allows the ICG to run as a standalone component capable of being called by the IDE and all the other components such as the PRC.

ICG will be able to produce, from a given DOML model, IaC code in multiple different target languages. The first version supports both Terraform and Ansible, and future versions may support further languages, possibly integrating new code generators as plugins.

In this first version, the generated code supports both provisioning and configuration and the second version will support orchestration. It allows provisioning Virtual Machines (VM) and other cloud resources for the selected Cloud provider and configuring those VMs with the installation of software components.

The internal components of ICG are shown in Figure 12 below.

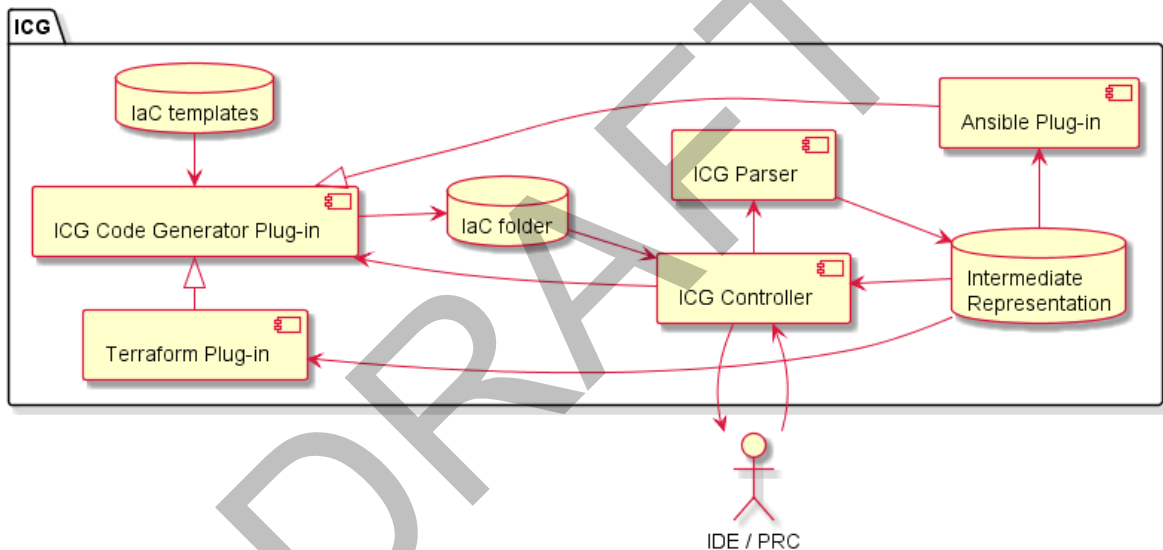


Figure 12: Internal ICG architecture

Component behavioural description

The sequence diagram shown in Figure 13 below exemplifies the behaviour of the ICG component from a high-level point of view. Internal interactions are shown in a summarized way; detailed interactions between the internal components are documented in the D3.4 deliverable.

As shown in the diagram in Figure 13, ICG is invoked through REST API. The ICG executable starts parsing the DOMLX model given as input. From the DOMLX model, ICG generates the Intermediate Representation, checks if any errors occur and generates IaC code using its templates.

Currently the ICG is called by the IDE at design time and the PRC at run time.

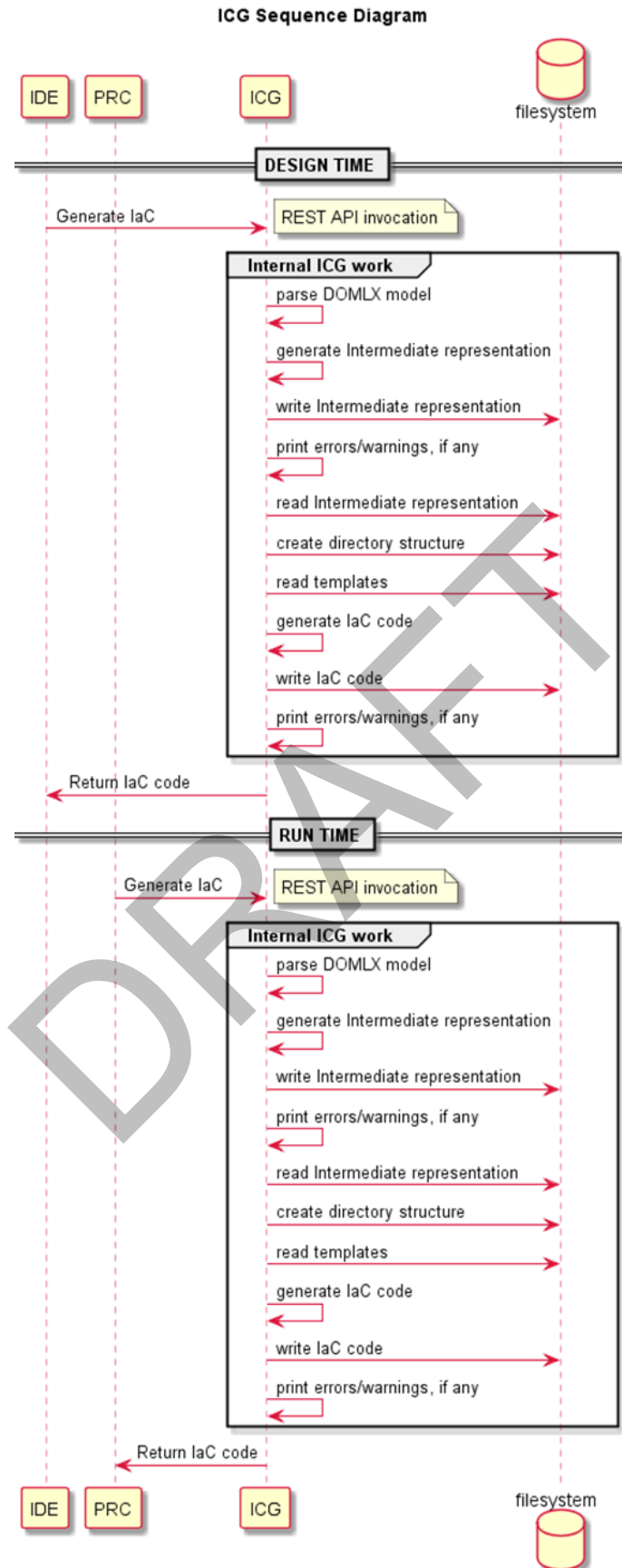


Figure 13: ICG internal and external behaviour

3.4.4 Verification Tool - VT

3.4.4.1 Model Checker (KR5)

Component Description

The Model Checker (MC) is the verification tool component which is devoted to check the main properties of the DOML model. In particular, the MC is going to check the consistency, completeness and some general issues of the model, including, if available, some special user-defined properties. The MC is called by the IDE, which sends to it a representation of the model, and returns either a positive result if the properties hold and no issues are found, or a negative result with some comprehensive counter-examples in case issues are indeed found.

Internally, the Model Checker consists of a component which translates the DOML model received from the IDE into an internal, logic-based format, that is called Target Logic Model Representation (or TLMR). The MC then calls the Logic Engine (LE) which is an external tool for the checking, that is the z3 SMT-solver (Satisfiability Modulo Theories). The output of the LE is then interpreted by the MC, in particular by the component that is called *Logic to DOML Mapper*, to translate the problems found by the LE (i.e., the counter-example) into a form compatible with DOML.

The internal architecture of the Model Checker is depicted in Figure 14.

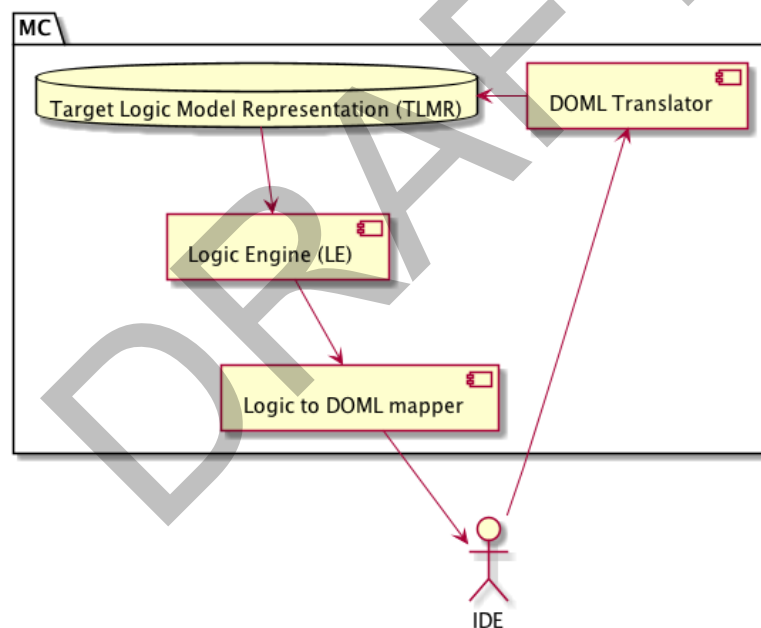


Figure 14: Internal architecture of the Model Checker.

Component behavioural description

Figure 15 below represents the typical behaviour of the MC. As depicted, the IDE sends a representation of the DOML model to the MC; then, the MC performs some abstractions or filtering, depending on the size, capabilities or other aspects of the model which could make the verification too expensive, for verification time or space needed.

The next step is the translation into the internal TLMR format and the verification of the standard consistency and completeness properties, by calling the external logic tool for the verification. In case more complex properties are present, these are translated into the TLMR format as well, and then the verification is performed.

The result of the verification is then returned to the IDE. In case of a negative verification result, the result also contains a counter-example evidencing the issues found by the MC.

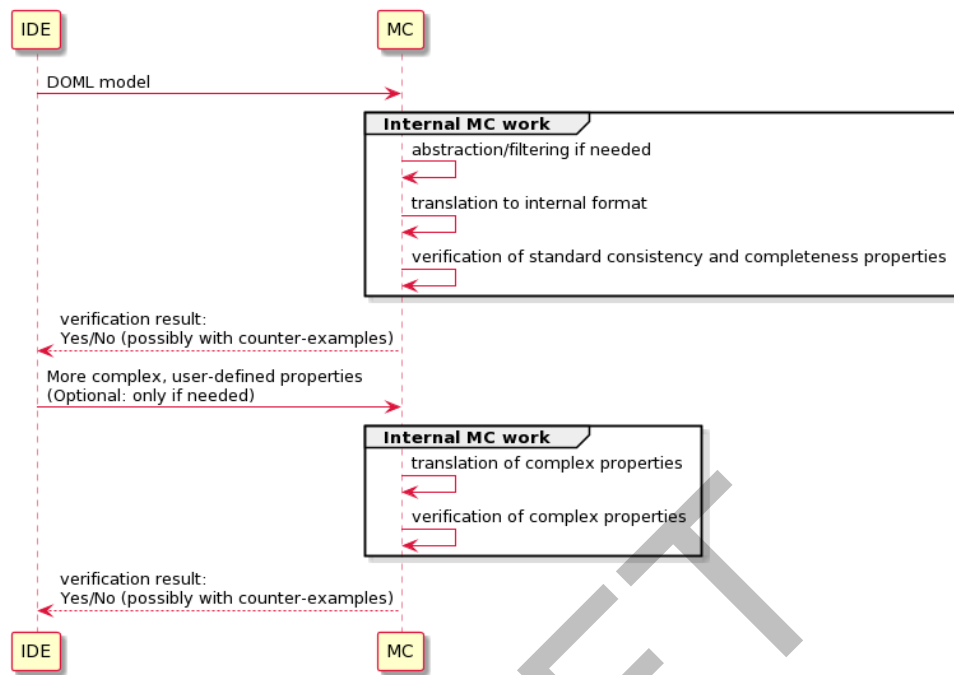


Figure 15: Model Checker internal and external behaviour.

3.4.4.2 IaC Scan Runner – (KR6 and KR7 executor)

As in depth presented in the D4.4 and D4.5 deliverables, the IaC Security Inspector (KR6) and Component Security Inspector (KR7) have very similar inputs and outputs handling. Therefore, a single tool has been designed in a way to act and fulfil the requirements of any or both KRs. IaC Scan Runner introduces the utilization of persistence layer for purpose of scan result persistence which enables browsing them later and also re-use of user-defined check scan preferences. In the Figure 16 the IaC Scan Runner sequence diagram is given.

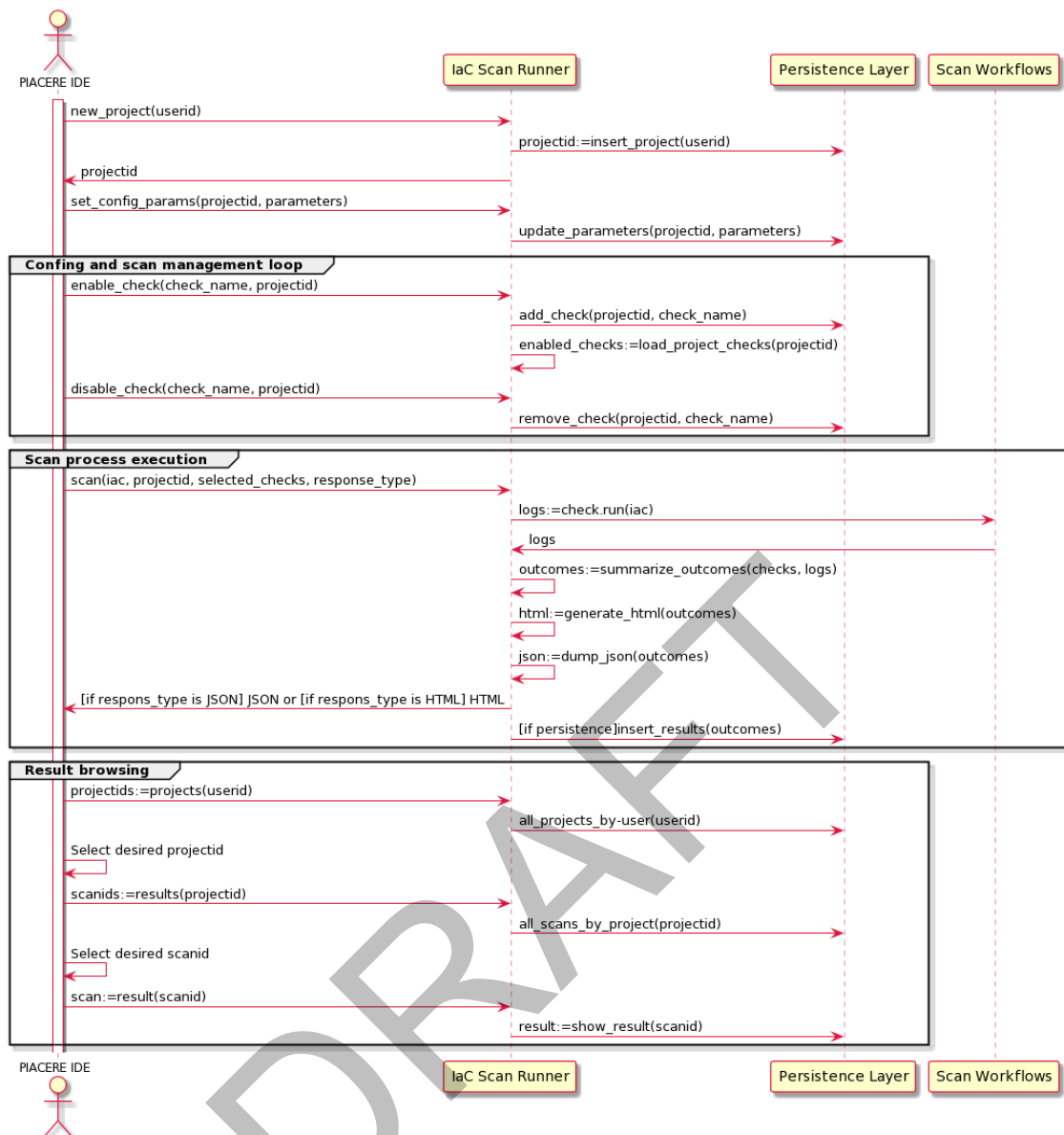


Figure 16: IaC Scan Runner archive scan workflow with persistence and configurations

In the first step of the sequence, it is possible to create a new scan project with default set of checks enabled. As outcome, project id is returned to the user/IDE. After that, user/IDE is able to set configuration parameters that would enable some of the tools (or additional paid services) that require client-related info, such as access tokens, password, identifiers or secrets.

The update of project parameters is persisted into database, so it can be later re-used and configuration time for each scan reduced.

Additionally, user/IDE is able to update the list of considered check tools, by enabling or disabling them individually, which is also persisted as part of project configuration.

Once the project configuration is done, user/IDE is able to assign their scan to a particular project in order to use the previously defined configuration. Additionally, list of preferred checks can be provided, so the final list of checks that will actually be performed is intersection of enabled and selected, considering file type and compatibility aspects as well. During the scan workflow, each of the check tools returns log. The returned logs are processed and summarized into JSON file.

Depending on the selected outcome, end-user receives either JSON file or HTML visualization summary which is generated taking the JSON file as input. Later, user is able to browse the previously created projects and scans.

3.4.4.3 *IaC Security Inspector (KR6)*

Component Description

IaC security inspector is the second verification tool providing statical analysis of the PIACERE designed application. In the contrast to the Model Checker, IaC Security Inspector performs security checks on the generated IaC instead of checking the DOML representation. The Security Inspector consists of a configuration part, where the set of security checks is selected and defined, and the runtime part, which performs checks on the IaC and builds the report.

The security inspector takes the IaC code generated by ICG from the DOML for an input and generates errors and recommendations. The first version of the IaC Security Inspector will focus on the framework, API and initial checks.

Component behavioural description

The IaC security inspector is an isolated service accessible to the other services through a REST API. The interface commands are very straightforward, facilitating the code inspection and configuration of the checks and are available through OpenAPI specification.

We defined performing the set of checks as a one scan of the IaC code. While designing the interfaces we realised that IaC Security Inspector and Component Security Inspector require the same interface performing different checks performed over the IaC. This led to the decision to create a single IaC *Scan Runner* component that will be able to run checks for both Inspectors. The detailed inspection of the checks showed us that some checks could be listed in both Inspector types (Figure 17).

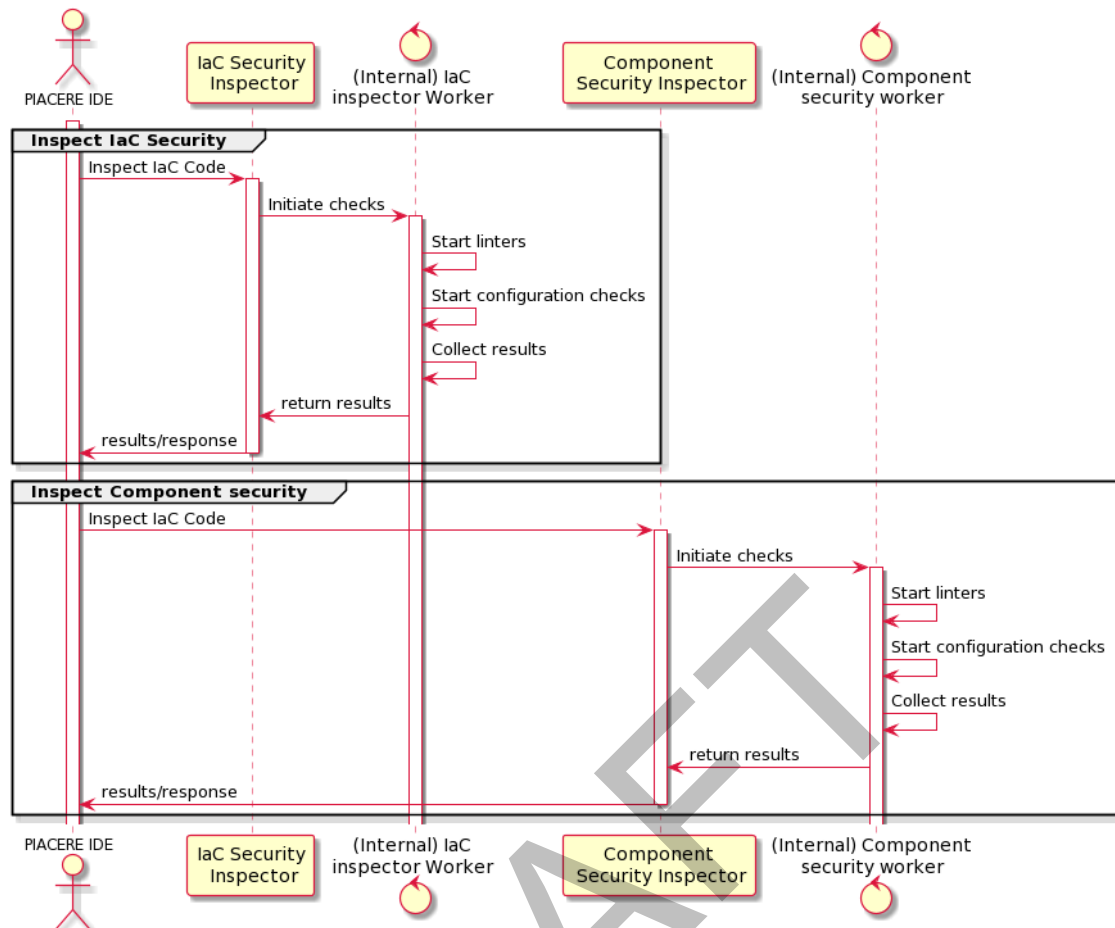


Figure 17: IaC Security and Component Security Inspector

3.4.4.4 Component Security Inspector (KR7)

Component Description

The component security inspector is a tool that makes static analysis of the IaC code, searching for components and searching for known vulnerabilities of the components. In other words, the Component Security Inspector will find dependencies used in the IaC and provide user a list of the vulnerabilities that are published by internet security authorities or are the result of misconfiguration of the component in IaC.

Component behavioural description

From the user's perspective, the behaviour of the Component Security Inspector will be identical to the behaviour of the IaC Security Inspector. The only difference is in the performed checks. As described in the section 3.4.4.2, the IaC *Scan Runner* component will run IaC Security Inspector or IaC Component Inspector checks included in scans.

3.4.5 IaC Executor Manager – IEM (KR10)

Component Description

The PIACERE project aspires to provide a common manner to utilize different IaC technologies in a unified way. The IaC component is of paramount importance to reach this overarching goal, since it oversees the utilization for the IaC code being generated in previous stages of the PIACERE infrastructure and execute the different technologies provided to obtain the desired infrastructural architecture. In addition, the IEM is able to leverage different IaC paradigms to

reach its goal, such as: the provisioning of the heterogeneous infrastructural devices required that may span different public and private cloud providers; the configuration of each and every infrastructural device that will support the PIACERE ecosystem, including the required dependencies for the elements that will support the PIACERE use cases; and the operationalization of the applications of the use cases that will utilize the PIACERE framework.

Additionally, the IEM will offer a unified approach to query the information regarding the deployments being made. This query method includes metrics about past and present executions of the IEM component, such as the duration of a given deployment or the status of the deployment (e.g., success, failure, pending). Furthermore, it provides a method to obtain information about the different IaC technologies supported by the IEM.

Finally, the IEM will expose its services through a REST API described in the OpenAPI specification format. This way, components willing to utilize the IEM, should implement its specification. The methods offered by the IEM must be used securely through token-based authentication technologies.

Component behavioural description

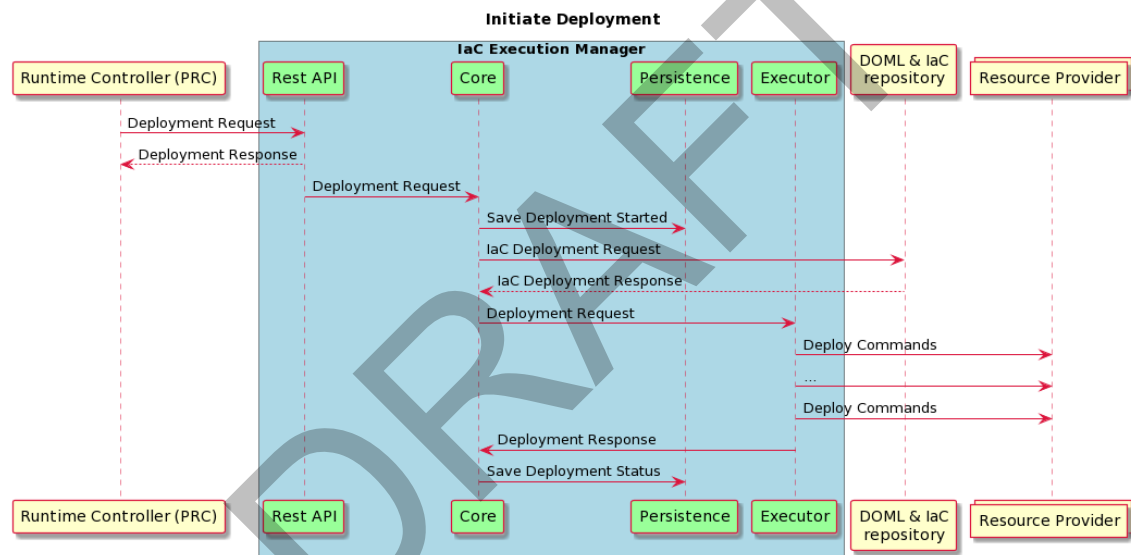


Figure 18: Start of deployment

The diagram above, Figure 18, exemplifies the sequence diagram regarding the start of a deployment. In this scenario, the Runtime Controller communicates with the IEM to initiate a deployment. This call is asynchronous given that an entire deployment may take a long time to finish, hence an immediate response is sent back to the Runtime Controller. Alongside the request, it provides the location of a deployment with the appropriate authentication and credentials. The IEM incorporates a persistence layer which will track the status of the recently started deployment. Then, the IEM retrieves the IaC files related with the initiated deployment and hands over the request to the executors, which will trigger the deployment in the Infrastructure Provider Resources. Finally, the status of the deployment is updated in the Persistence layer so it can be queried appropriately.

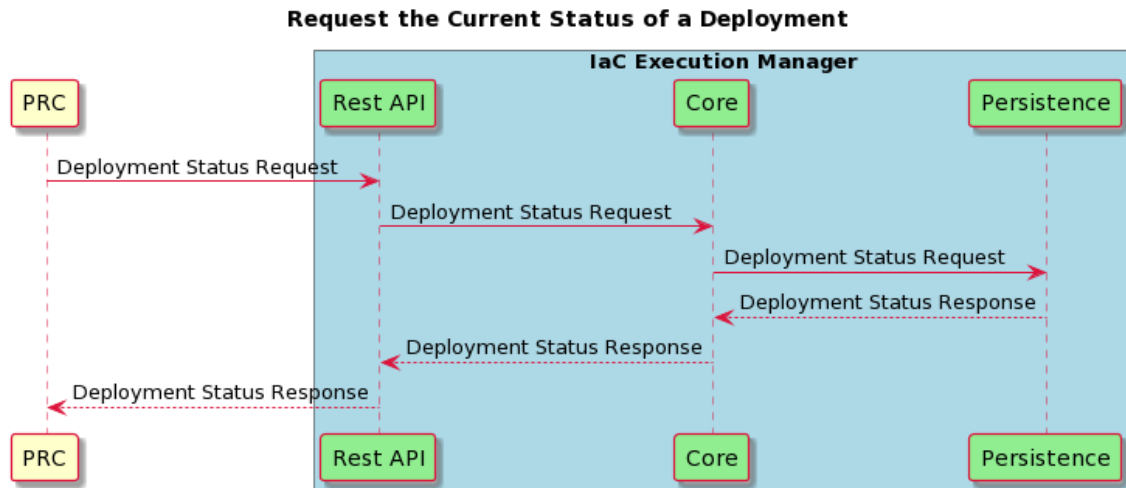


Figure 19: Request of the status of a deployment

The diagram above, Figure 19, shows the sequence diagram regarding the request of the status of a deployment by the user. This is a synchronous call; hence the user obtains real time feedback on the request. The IEM stores this information in the persistence layer and keeps track of all the present and past deployments that have been initiated by this component.

3.4.6 Runtime Controller – PRC

Component Description

PIACERE Runtime Controller (PRC) is the main control component of PIACERE runtime. It is a workflow engine that guides the overall workflow within the PIACERE runtime. Actions of PRC are targeted against a specified set of resource providers (including Canary and Production) via the integrated components such as the IEM (IaC Executor Manager) and the IA (Infrastructure Advisor), particularly its own controller component.

The PRC is involved in the PIACERE framework integration. This is described in more detail in a later section of this document.

Component behavioural description

PRC does not have any sequence diagrams as there is no native sequence diagram to be shown. PRC integrates the flows of other components into a single, coherent flow spanning the whole PIACERE runtime. The IDE queries and controls the PIACERE runtime via the PRC.

3.4.7 Canary Sandbox Environment – CSE (KR8)

3.4.7.1 Canary Sandbox Environment Provisioner - CSEP

Component Description

The role of the Canary Sandbox Environment Provisioner (CSEP) is to create the desired Canary Resource Provider (CRP). This may entail provisioning and configuring new systems to provide the desired platform. There are two approaches to the CSE: to provide a real (non-simulated) CRP and a simulated one. Depending on the variant, the scope and characteristics of testing differs. Real providers require resources and allow to complete all steps of deployment as long as the supporting infrastructure (beneath the created CRP) is sufficient. The assumption is that the user is able to provide the hardware (e.g., because they have bare metal or virtual machines, either on premise or elsewhere – the CSE is agnostic to that). On the other hand, the simulated

variant does not consume resources but does not allow further steps other than provisioning of the infrastructure elements. The initial set of planned supported platforms is OpenStack (for real [non-simulated] actions) and the Canary Sandbox Environment Mocklord (for simulation).

Component behavioural description

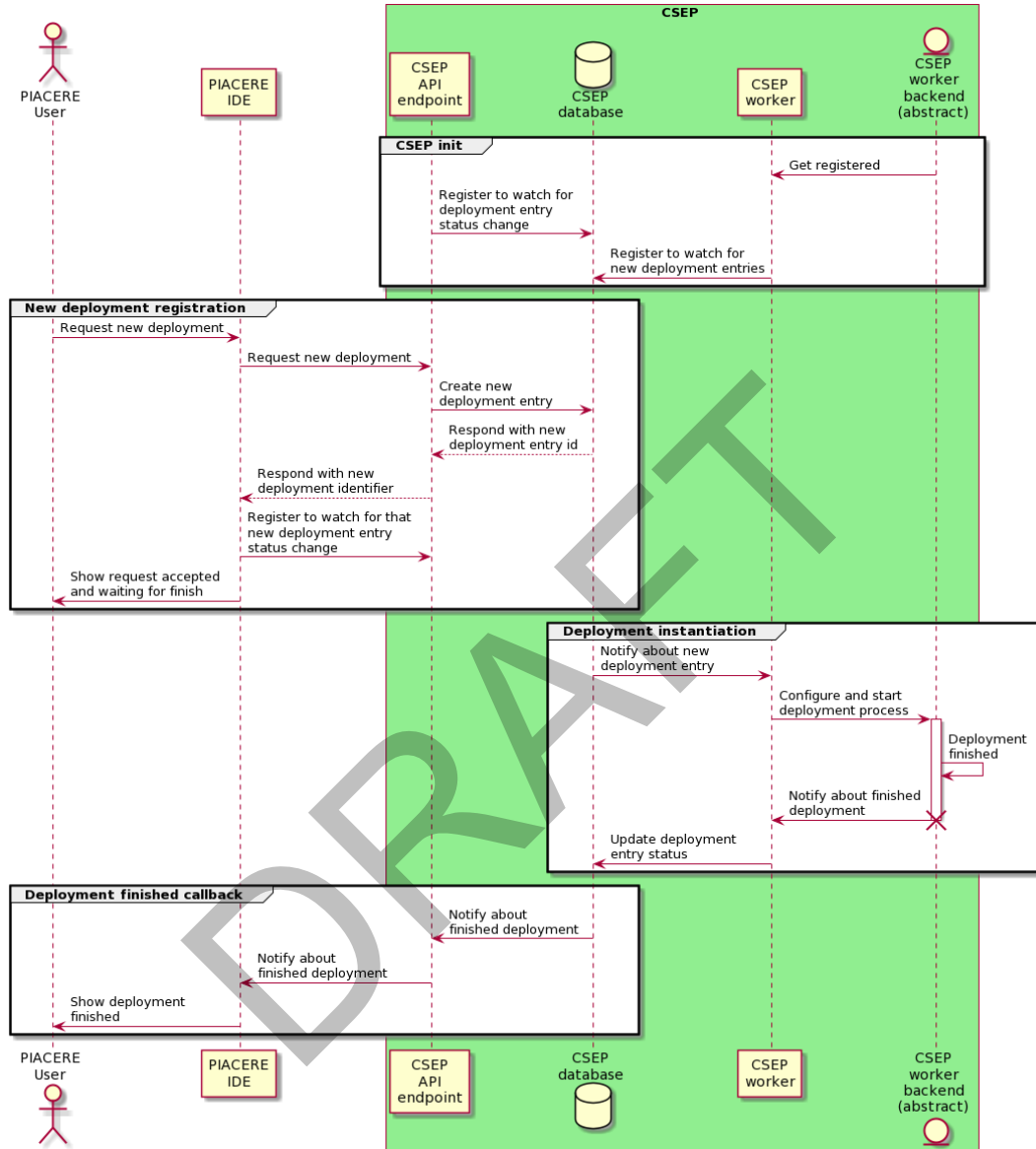


Figure 20: Canary Sandbox Environment Provisioner (CSEP)

As shown in Figure 20 In the initialisation stage, both the API and worker components connect to the internal database to watch for deployment status changes.

The primary sequence of actions regarding the Canary Sandbox Environment Provisioner (CSEP) involves provisioning of the chosen Canary Resource Provider (CRP) that can be used as a Resource Provider (RP) with other PIACERE tools, notably IEM. The user, possibly indirectly via the IDE, invokes the command to provision a new CRP (create new deployment). The CSEP API component handles this request and creates an appropriate record in the internal database. This record is then detected by the worker component and acted upon (and updated in the internal database along the way). Finally, when the worker finishes its job, i.e., deploys the CRP or fails

to do so, the worker saves the final state in the internal database. This information can then be read by the user, possibly indirectly with IDE.

The alternative and complementary flows involve the following actions:

- destroying the deployment (when the flow of actions is analogous to creation)
- listing deployments
- getting details about a particular deployment.

3.4.7.2 Canary Sandbox Environment Mocklord – CSEM

Component Description

Canary Sandbox Environment Mocklord (CSEM) is to be provisioned on demand by the CSEP. The role of CSEM is to simulate an existing resource provider so that the user can easily test interactions against it. The plan is to research the usefulness of such approach to dynamic IaC testing. The prototype will target a subset of AWS [3] APIs. CSEM is assumed to have a much lower cost compared to real (non-simulated) resource providers. Due to simulation, this variant of Canary Resource Provider will allow only the provisioning step to happen.

Component behavioural description

CSEM does not have any sequence diagrams as there is no native sequence diagram to be shown. CSEM will offer a simulation of the possible upstream API flows, e.g., actions possible against the EC2 API of AWS.

3.4.8 Infrastructure Advisor

3.4.8.1 IaC Optimizer Platform - IOP (KR9)

Component Description

The optimization problem formulated in PIACERE and solved by the IOP consists of having a service to be deployed and a catalogue of infrastructural elements, with the main challenge of finding an optimized deployment configuration of the IaC on the appropriate infrastructural elements that best meet the predefined constraints (e.g., types of infrastructural elements, technical requirements, and so on). In this context, it is the IOP component which is the responsible for finding the best possible infrastructure given the input data received. This input data is provided in DOML format and will include the optimization objectives (such as the cost, performance, or availability), optimization requirements, and previous deployments (in case it is necessary). Then, the IOP performs the matchmaking for the infrastructure via the execution of optimization intelligent techniques by using the information taken as input against the available infrastructure and historical data, available from the catalogue of Infrastructural elements.

In other words, the optimizer will use artificial intelligence optimization algorithms, seeking for an optimized deployment configuration of the IaC on the appropriate infrastructural elements that best meet the predefined constraints. Thus, the IOP will success if it is able to propose the most optimized deployment configuration of the infrastructural code taking into consideration the constraints predefined. To this end, several deployment configurations will be shown and ranked.

Finally, two considerations should be considered. The first one is that the problem to be optimized will be a multi-objective one, which means that it will be composed by several conflicting objectives (such as cost and performance). The second aspect to consider is that two different optimizations will be conducted in the context of PIACERE: the initial deployment of

the service, and the redeployment of an already running service (if the Self-Healing detects it is necessary).

Component behavioural description

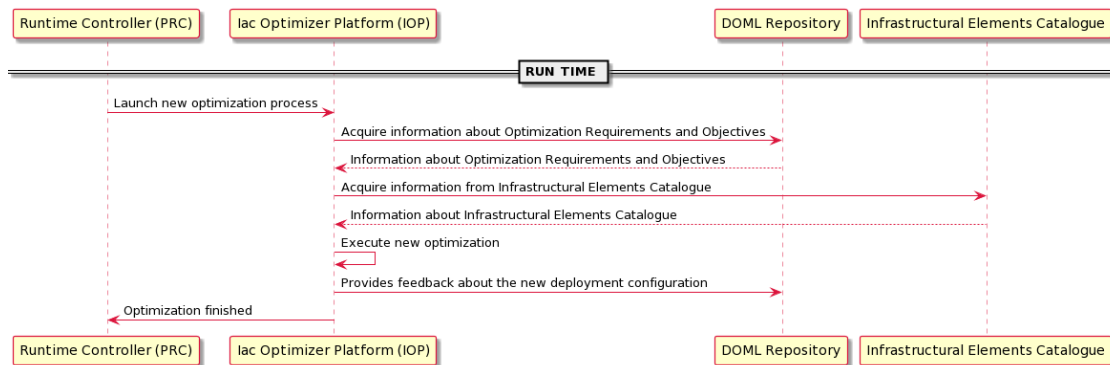


Figure 21: IOP in Run Time

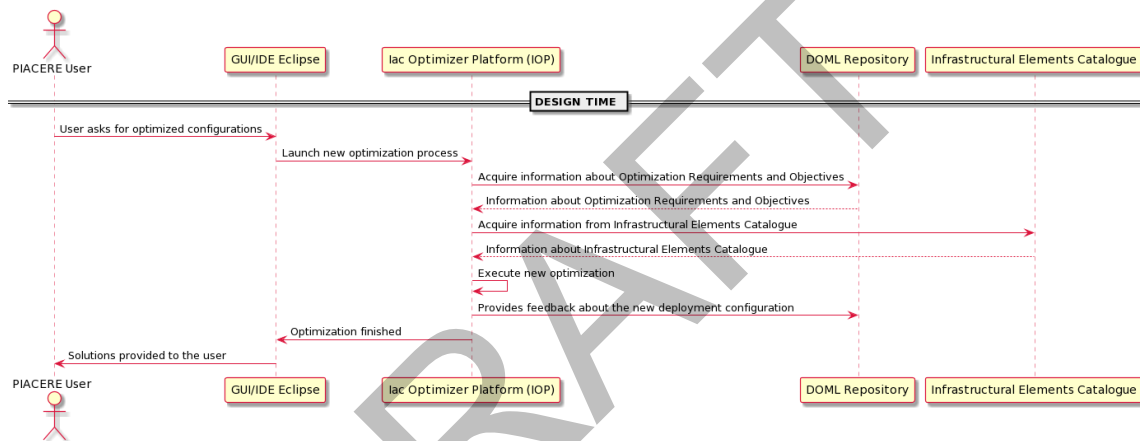


Figure 22: IOP in Design Time

The IOP receives all the information about the available elements from the IEC. These elements are transformed in optimization variables and are employed for optimizing the functional and non-functional requirements. Obtained optimized solutions are provided to the Runtime Controller when the service is called if the self-learning has detected the necessity of finding a new deployment configuration. Furthermore, the IOP can also be called in the Design Time phase, when the first deployment is needed. In this situation, the user can ask for the optimization of the deployment through the PIACERE IDE.

Having said that, the role of the IOP in both Run Time and Design Phase can be summarized as follows:

- *Role of the IOP in the Run Time phase* (Figure 21): in this phase, the IOP is involved in the self-healing process when needed asking for a re-deployment, and it is called by the PIACER Runtime Controller.
- *Role of the IOP in the Design Time Phase* (Figure 22): in this case, the IDE can call the IOP as an optional step (user choice), after the DOML Model Checker interaction. Thus, the IOP will return the optimization to the IDE, and the user can accept or not the proposed optimization.

3.4.8.2 *Run-time Monitoring System (KR12)*

Component Description

The monitoring mechanisms present in PIACERE allow gathering non-functional measures over the infrastructure resources that run the components that build up the application. Currently PIACERE supports the monitoring of two non-functional measures categories: performance and security.

- The **Performance Monitoring** component focuses on gathering performance related measures from the infrastructure resources. The measures are gathered by agents running in the infrastructure resources: this allow us to gather data about some individual metrics that may be useful to get idea about the overall health of those resources. Examples of metrics are: memory use, disk use, processes, CPU usage, etc.
- The **Security Monitoring** component focuses on gathering performance related measures from the infrastructure resources. The measures are gathered by agents running in the infrastructure resources.

Component behavioural description

There are two main aspects in the lifecycle of the Monitoring components: the new applications configuration and the monitoring loop.

The new application configuration has two main parts: the deployment of monitoring agents and the configuration of the monitoring components to follow the deployed application. The deployment of the monitoring agents is expected to be done during the application deployment as part of the activities requested to by the PRC to the IEM. The configuration of monitoring components to follow the deployed application is centralized by the monitoring Controller. This is a utility component in charge of notifying the inner monitoring components to start and stop gathering information for a given application. This is shown in the Figure 23 under the group “start”.

On the other hand, the application decommissioning has also two main parts: the un-deployment of the agents and the configuration of the monitoring components to stop following the deployed application. As with the new application configuration, the un-deployment of the monitoring agents is expected to be done during the application decommissioning as part of the activities requested by the PRC to the IEM. The configuration of the monitoring components to stop following the deployed application is centralized by the monitoring Controller. This is shown in the Figure 23 under the group “end”.

The **Performance Monitoring** component focuses on continuously gather the data coming from the multiple monitoring agents, evaluate the configured threshold and if necessary, send alerts to the Self-healing component. This is shown in the Figure 23 under the group “loop”.

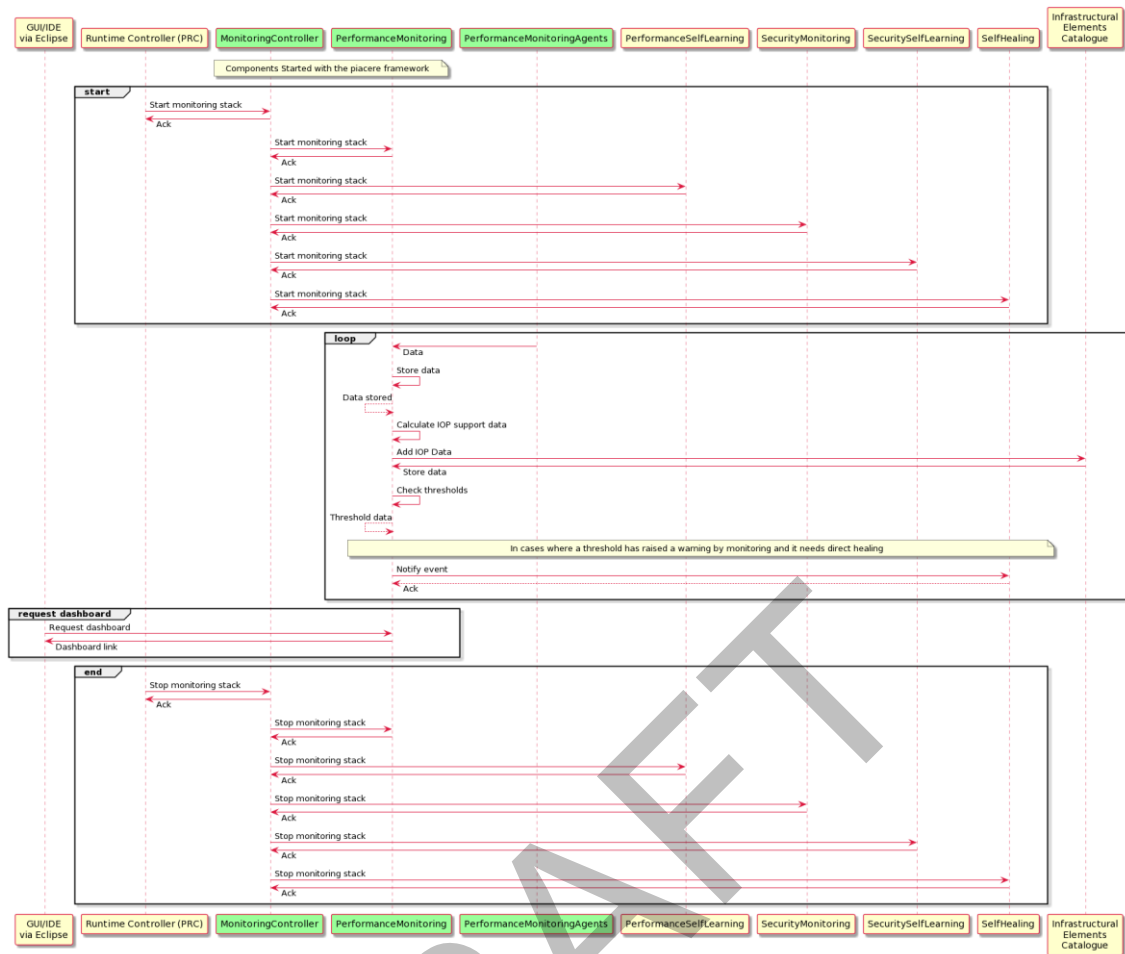


Figure 23: Monitoring

NOTE: The above figure does not cover the security monitoring activities as these are covered in the security monitoring diagram (Figure 24).

The **Security Monitoring** component's role is two-fold: to gather data from the security monitoring agents and notify the Self-Healing component on the potential issues to be acted upon, and to gather data for the Security Self-Learning component for detecting anomalies regarding security events. The monitoring system is depicted in Figure 24. As soon the application has been configured and deployed with the rest of monitoring infrastructure, the data is started to be gathered and analysed. Events are being continuously evaluated and in case an event related to a specific PIACERE-relevant metric and with the PIACERE rule being triggered, the Self-Healing component is being notified on this event.

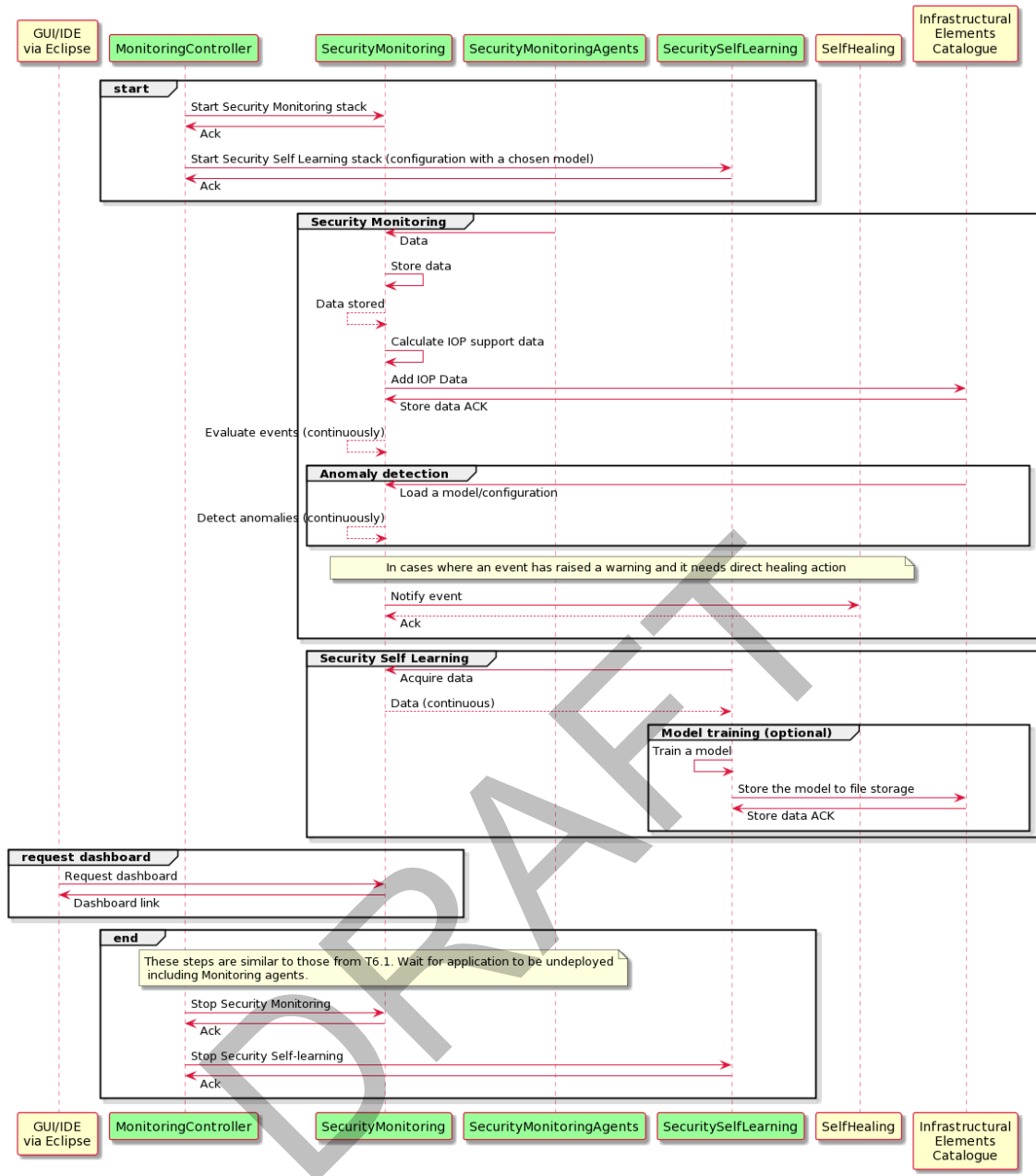


Figure 24: Monitoring System

Note: Security Monitoring component is depicted above the Security Self-Learning part.

3.4.8.3 Self-Learning (KR11)

Component Description

The self-learning mechanism present in PIACERE allows analysing the deployed elements using a set of monitored parameters and predicting anomalous situation that would require an action (i.e., deployment of new infrastructural elements). The Self-Learning component will be responsible for checking that the different elements present on the platform are in good condition and do not show any degraded or anomalous behaviour. Currently, PIACERE supports the self-learning of two non-functional categories: performance and security.

- The **PerformanceSelfLearning** component focuses on incrementally online learning and predicting the performance of the elements to guarantee their constant high-level

performance. To do that, the component receives monitoring data from the **PerformanceMonitoring** component.

- The **SecuritySelfLearning** component makes use of state-of-the-art Natural Language Processing (NLP) architectures to model log streams as a language and capture their normal operating conditions. These models can then be used to detect deviations from the normal behaviour.

Component behavioural description

The **PerformanceMonitoring** provides the **PerformanceSelfLearning** component with the monitoring data of each element hosted in the system, after being requested by the **PerformanceSelfLearning**. The CPU usage idle, as part of this monitoring data, is requested, learnt, and predicted in an online fashion manner by the **PerformanceSelfLearning**, through an online learning algorithm that can deal with drifts and anomalies. When the prediction of the next CPU usage idle data point is below a threshold (i.e., 70%), the **PerformanceSelfLearning** component sends a warning to the **Self-Healing** component. Then, this latter component will have to decide what to do or how to consider such warning. This is shown in the Figure 25.

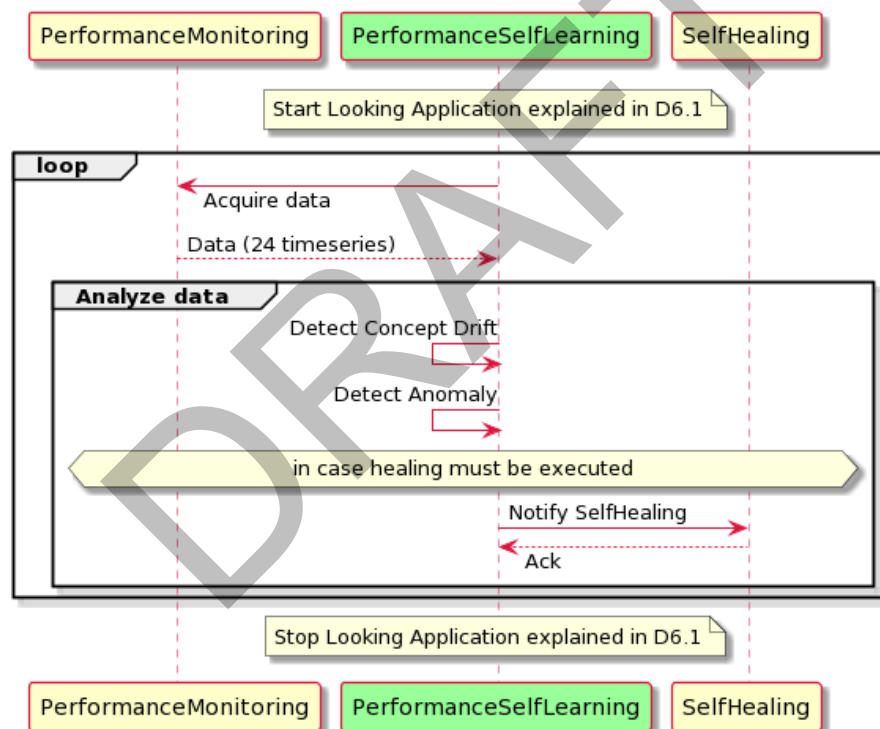


Figure 25: Self-Learning (Performance)

NOTE: The above figure does not cover the security monitoring activities as these are covered in the security monitoring diagram (Figure 24).

The **SecuritySelfLearning** component (activity diagram depicted in Figure 26) receives data from the **SecurityMonitoring** component. As a first necessary step, a specified subset of the data has to be used to train a behavioural model. This subset of data, along with the necessary configuration files, is provided to the **ModelTraining** component, which eventually stores every trained model in the **ModelRepository**. Once a model is trained, this step is repeated only if requested to do so. A trained model is loaded from the **ModelRepository** to carry out anomaly

detection of the data received from the **SecurityMonitoring** component. Under previously specified conditions e.g., high number of anomalies in a short period, the **SecuritySelfLearning** component will notify the **Self-Healing** component.

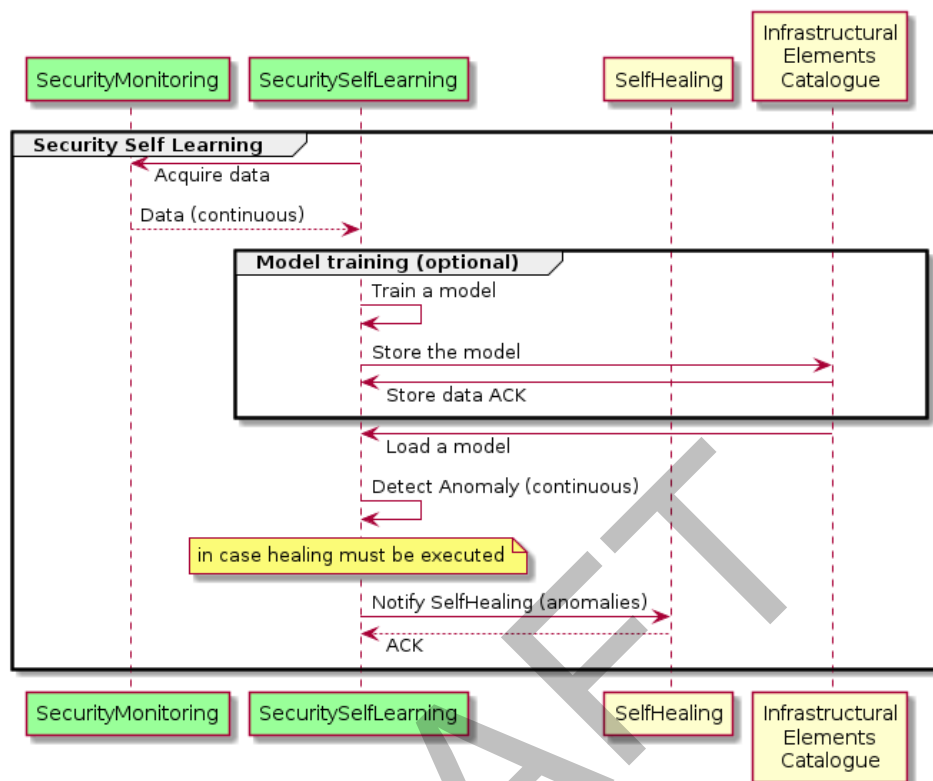


Figure 26: Security Self-Learning

3.4.8.4 Self-Healing (KR11)

Component Description

The Self-Healing mechanism present in PIACERE allows to receive incidence or forecast notification from monitoring and self-learning components. Based on the typology of the notification the Self-Healing component identifies the mitigation strategy to be applied and proceeds with its execution.

Component behavioural description

The SelfHealing component waits for alerts from the monitoring components. This includes: Performance monitoring, Performance Self-Learning, security monitoring and Security Self-Learning. There will be different types of alerts for example monitoring components will inform that some threshold has been exceeded or that something has happen, while Self-Learning components will inform that something may happen based on the evolution of the metrics analysed.

Once a notification has been received, the Self-Healing component classifies the event and based on that classification it applies a strategy. The strategy will be realized by sending a Self-Healing workflow to the PRC. Different strategies are envisioned, such as reboot, migrate, scale. This is shown in Figure 27.

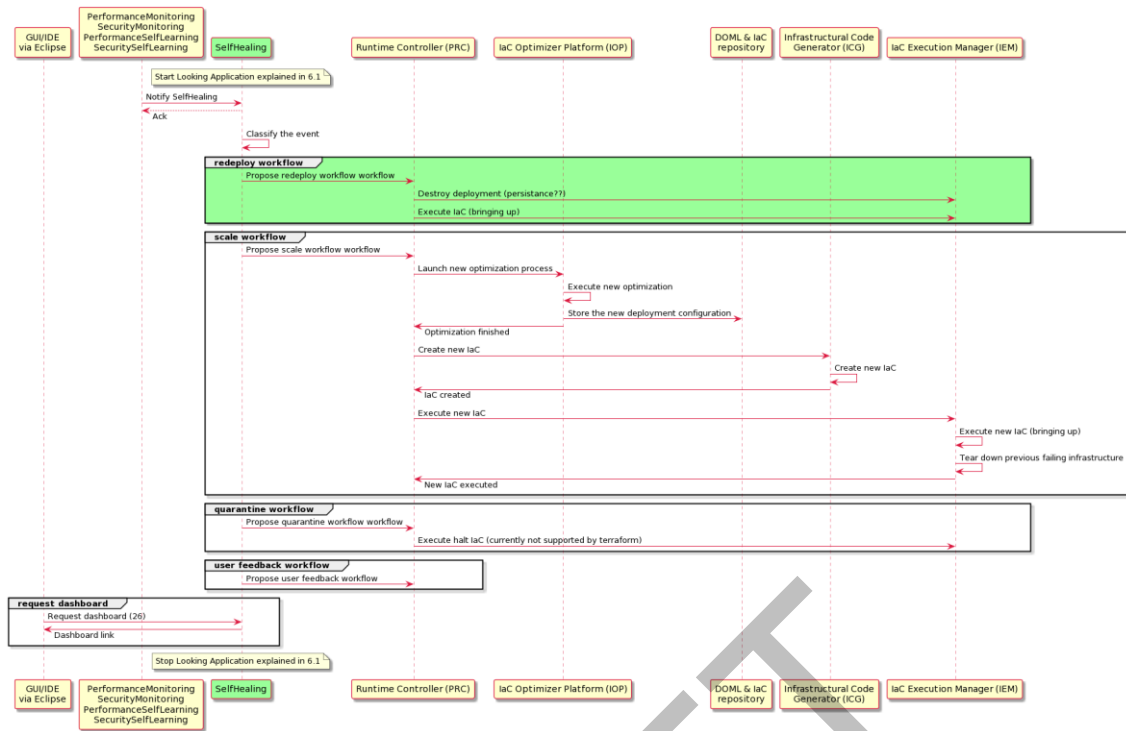


Figure 27: Self-Healing

3.4.9 Infrastructural Elements Catalogue (KR9)

Component Description

The Infrastructural Elements Catalogue present in PIACERE stores information about the services available at service providers as well as the instances of each of these services being used by the different application being deployed by the PIACERE infrastructure.

Component behavioural description

The Infrastructural Elements Catalogue component is a persistence component that stores information required by different PIACERE components. As a persistence component there are two critical aspects to be covered: how the information is added and how the information is retrieved.

Regarding the feed of information there are three main interactions: the GUI/IDE (Eclipse), the PRC and the monitoring components. The GUI/IDE will add information about the available services. The PRC will add information about the instances used from those available services. Finally, the monitoring components (both performance and security) will add average information that will be latter used by the IOP. This is shown in Figure 28.

Regarding the usage of information, there is one main interaction: the IOP. The IOP requires to use information about the services in order to identify the optimal combination of services to support the application non-functional requirements. This is shown in Figure 28.

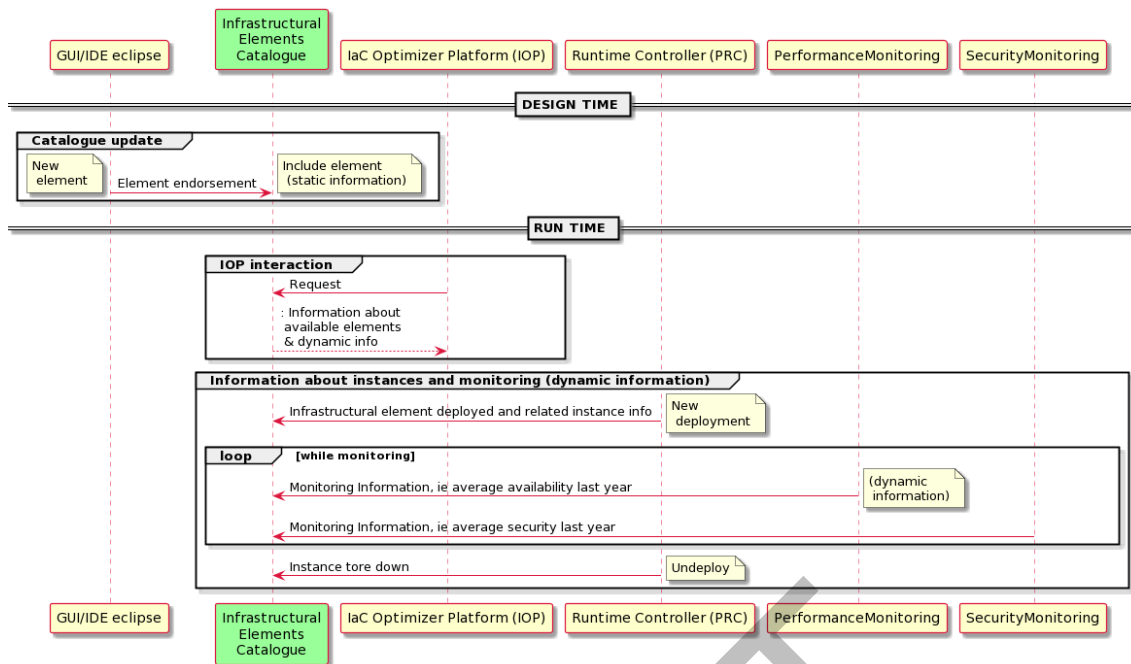


Figure 28: Infrastructure Elements Catalogue

3.5 PIACERE Multi-User Approach

The purpose of this chapter is to evaluate which components may need to manage multiple users with different roles or privileges, which ones are ready or have defined the steps to accommodate this functionality and, which ones should be analysed more to manage the multi-user approach. The result of this analysis is presented in Table 11, where the **KR** column corresponds to the key result, **Component** to the PIACERE component name, **Phase** to the stage where the component is used, **Multi-user** indicates if the component may need this feature, **Ready** presents the status of feature’s achievement, **Notes** includes some comments related to the status and the need of more investigation.

Although the multi-user scenario is not necessary for the realization of the use cases and therefore it is not essential for the realization of the PIACERE prototype (there are no requirements related to the need to have multiple users with different roles or privileges), it could be connected to the exploitation work-package for future evolutions of the PIACERE framework, as it is an advanced feature relevant in the industrial context.

Table 11: Piacere multi-user approach

KR	Component	Phase	Multi-user	Ready	Notes
KR1	DevSecOps Modelling Language (DOML)	design	N/A	N/A	This is not a component but the modelling language.
KR2	Integrated Development Environment (IDE)	design	Yes	Yes	The IDE will be a desktop tool that will use a repository to store the models (git). The models will be synchronized in a similar way that the code (developer projects) is synchronized. The user of the IDE should commit the model before it will be available for other users or tools.
KR2	DOML & Repository	design	Yes	Yes	The IDE supports Git repositories.

KR	Component	Phase	Multi-user	Ready	Notes
KR3	Infrastructural Code Generator (ICG)	design & runtime	No	N/A	ICG by itself is not impacted by a multi-user scenario: like any compiler it depends on its input and generates the related output. The execution of the generated IaC code may be impacted in some cases, e.g. Terraform should use a remote state common to all developers.
KR4	DOML Extension mechanism (DOML-E)	design	N/A	N/A	This is not a component but the modelling language.
KR5	Verification Tool - Model Checker	design	No	N/A	This component is activated by the IDE only to check the validity of the models used for the DOML code.
KR6	Verification Tool - IaC Security Inspector	design	Yes	No	This service can work independently, executed on demand individually by all team members. The only issue arises if the output needs to be presented to all developers, independently of who did run the service. It should be analysed more.
KR7	Verification Tool - Component Security Inspector	design	Yes	No	This service can work independently, executed on demand individually by all team members. The only issue arises if the output needs to be presented to all developers, independently of who did run the service - so in this case we need to store outputs in the user/team space. It should be analysed more.
KR8	Canary Sandbox Environment Provisioner (CSEP)	test	Yes	Partially	This could benefit from PIACERE-wide multiuser identity and permissions to let user deploy independently (or shared on demand). It should be analysed more.
KR8	Canary Sandbox Environment Mocklord (CSEM)	test	Yes	Partially	This can be deployed once per each user and avoid any collisions. It should be analysed more.
KR9	IaC Optimizer Platform (IOP)	design & runtime	Yes	No	This service can work independently, and it is executed on demand. This service can be called under two different contexts: for the first deployment of the service and when it is required by the self-healing mechanism. Because this service is executed independently regardless its context, it does not create any collision. It should be analysed more.
KR9	Infrastructural Elements Catalogue	design & runtime	Yes	Yes	The catalogue relating to the infrastructures used by the components of the PIACERE framework are stored on this git repository.
KR10	IaC Executor Manager (IEM)	runtime	Yes	No	Each deployment can be shared among different users. This component should not be affected by this multi-user scenario. It should be analysed more.
KR11	Infrastructure Advisor (IA) - Self Learning - Performance Self Learning - Security Self Learning	runtime	Under evaluation	No	This component is independent from the multi-user scenario. Monitoring tool calls self-learning component at the beginning of the monitoring process, and the self-learning remains alive/working during the lifespan of the process. There is no human interaction

KR	Component	Phase	Multi-user	Ready	Notes
KR11	Infrastructure Advisor (IA) - Self Healing	runtime	Under evaluation	No	with this component, at least for the moment. It should be analysed more in the future as monitoring data and components as self-learning/self-healing will probably need some API or GUI (Dashboard) so the user will be able to see the logs and results of that self-healing.
KR12	Infrastructure Advisor (IA): - Monitoring System - Performance Monitoring - Security Monitoring	runtime	Under evaluation	No	This component is affected in the same way as runtime monitoring system. However, each deployment of the system can be shared among different users (users of the group working on the same project). The component should not be affected by the multi-developers scenario. Access to the data originating from the component needs to be managed by monitoring or some other component. The component itself will be using system-level user (or the user from the configuration) to manage/access component's data directly. It should be analysed more.
KR13	Runtime Controller (PRC)	runtime	Yes	No	This component integrates others and therefore is not directly impacted by single vs multiple user scenario. Other components would have to be multiuser-aware. It should be analysed more.

3.6 PIACERE Security Approach

Regarding Security approach there are different perspectives:

- the trustworthiness and security of the IaC and of the associated software components, realized with the PIACERE Verification Tools 3.4.4,
- the continuous monitor of performance and security metrics gathering security-related measures from the infrastructure resources, realized with the PIACERE security monitoring components 3.4.8,
- the secure communication between the components of PIACERE Framework.

The realization of secure communication between components is related to requirement 10, described in Table 4: *“The communication within the different components of the architecture should be done in a secure way”*.

At the current state of the art, a secure communication between the components of the PIACERE framework is ensured by the use of REST APIs. As regards the management of the credentials necessary to access internal services (e.g. the DOML code repository and internal catalogs such as the IEC), the Json Web Token (JWT) credential transmission mechanism is used. To safeguard the credentials the use of "Hashicorp Vault" has been suggested and used in the CSEP component). Details of how these tools have been integrated into the KRs are available in the related deliverables.

The communication between components in a safe way is continuously updated during the integration process still on going.

3.7 PIACERE Scenarios

The purpose of this chapter is to describe some scenarios of the using of PIACERE framework from the user perspective, linking them to the requirements according to the following needs:

- as a PIACERE user I want prepare and use the PIACERE IDE environment to design applications and verify them,
- as a PIACERE user I want to test, deploy and manage the lifecycle of the deployment.

The first need is related to the *design time* that focuses on the tools needed to design, plan, create, verify the trustworthiness of IaC as well packing it for the deployment. The second one is related to the *runtime* that focuses on the tools needed to package, release and configure IaC as well to monitor the infrastructure.

In Table 12 the main PIACERE scenarios are presented, using the Cucumber/Gherkin notation [4].

This table shows the relationship between the scenarios proposed to showcase the PIACERE framework and the requirements met for their realization. Using the proposed scenarios to test the PIACERE framework, it will be possible to verify if the requirements are met. More detailed scenarios are provided by the KRs in the specific deliverables, covering the specificities of the usage of each KR.

Table 12: Piacere Scenarios

Phase	Title	Scenario	Main KR	Description	Requirements
design-time	Create new DOML	Given an installed PIACERE IDE When user starts a new PIACERE DevOps project Then a new DOML file is created	KR2	From IDE GUI the user can create a new project and a new application design (DOML file).	REQ28, REQ40, REQ41, REQ42, REQ43, REQ44, REQ62, REQ64
design-time	DOML Development	Given a DOML document When user modifies the DOML content And creates a new DOML node or changes the DOML content Then autocompletion functions helps the user to modify the content	KR1	From IDE GUI the user can create a new DOML node or change the DOML content.	REQ01, REQ25, REQ26, REQ27, REQ28, REQ29, REQ30, REQ58, REQ59, REQ61, REQ62, REQ63, REQ70, REQ76
design-time	Convert DOML to DOMLX	Given a DOML document When a user navigates to the DOML document And right-clicks on it And selects "Piacere" And selects "Generate DOMLX Model" Then a DOMLX file containing the original DOML document in XML format is generated	KR1	The user can create from IDE a different representation of DOML.	REQ01, REQ 25, REQ 26, REQ27, REQ28, REQ29, REQ30, REQ58, REQ59, REQ61, REQ62, REQ63, REQ70, REQ76

Phase	Title	Scenario	Main KR	Description	Requirements
design-time	DOML Verification - Model Checker	Given A DOMLX document And a check configuration is prepared When a user navigates to the DOMLX document And right-clicks on it And selects "Piacere" And selects "Validate DOML" Then a KR5 model checker is invoked And a response is returned	KR5	The user can check inconsistencies and validate the DOML through the model checker.	REQ68, REQ69, REQ71, REQ95
design-time/ runtime	Optimise IaC	Given a verified DOML document And the user has already introduced the optimization objectives When user navigates to the DOML document, And right-click on the file And selects "Optimise" Then the IOP is invoked And runs the optimisation algorithm And returns the optimised IaC examples And the user evaluates and verifies the output results	KR9	The user can run the optimisation algorithm inserting the objectives to optimize and the requirements that should be met by the solution.	REQ03, REQ04, REQ46, REQ98
design-time	Initiate ICG	Given a verified DOMLX document When user navigates to the DOMLX document, And right-click on the file And selects "Piacere" And selects "Generate IaC Code" Then ICG is invoked And generated IaC is returned	KR3	The user can generate IaC code invoking the ICG that reads DOMLx.	REQ31, REQ77, REQ96, REQ100
design-time	Initiate IaC Scan runner	Given a generated IaC code from DOML And a set of required checks is enabled on IaC Scan Runner When user navigates to the IaC document/zip, And right-click on the file And selects "IaC Scan run" Then IaC scan runner is invoked And a response is returned	KR6, KR7	The user can perform security scan invoking IaC Security Inspector and/or Security Component Inspection. The scan result is returned to the user.	REQ24, REQ65, REQ66, REQ91
runtime	Initiate the deployment of the Canary Sandbox Environment (CSE)	Given the PIACERE IDE And the Canary Sandbox Environment Provisioner (CSEP) connection details When The user provides the CSEP connection details to the IDE And user requests CSE creation from the IDE Then The user is notified that the CSE deployment request is accepted And the new deployment record appears in the list of CSE deployments in the IDE	KR8	The user can invoke to initiate the deployment of the desired Canary environment specifying the resource provider connection details in a user-friendly form.	REQ33, REQ34, REQ37, REQ38, REQ39

Phase	Title	Scenario	Main KR	Description	Requirements
runtime	Check the status of CSE deployment	Given the PIACERE IDE And an initiated CSE deployment When the user navigates to the CSE deployment record in the IDE And the user requests more details about the CSE deployment Then the IDE presents the user with CSE deployment status details	KR8	The user can check the status of the deployment of the desired Canary environment.	REQ33, REQ34, REQ37, REQ38, REQ39
runtime	Initiate the deployment of IaC on CSE	Given the PIACERE IDE And a deployed CSE And a generated IaC When the user navigates to the IaC code documents And the user selects "Run in PIACERE Canary Sandbox Environment" And the user selects the CSE to use Then the user is notified that the deployment is initiated And the new deployment record appears in the list of IaC deployments in the IDE	KR10, KR13	The user can deploy IaC in the Canary Sandbox environment already deployed. In this scenario, the IDE calls the PRC which, in turn, calls the IEM.	REQ10, REQ12, REQ 55, REQ 81, REQ82, REQ83, REQ84, REQ85, REQ87, REQ88
runtime	Initiate the deployment of IaC in the target environment	Given the PIACERE IDE And a generated IaC And the target environment credentials When the user navigates to the IaC code documents And the user selects "Run in the target environment" And the user chooses the target environment credentials Then the user is notified that the deployment is initiated And the new deployment record appears in the list of IaC deployments in the IDE	KR10, KR13	The user can deploy IaC in the target environment. In this scenario, the IDE calls the PRC which, in turn, calls the IEM.	REQ10, REQ12, REQ 55, REQ 81, REQ82, REQ83, REQ84, REQ85, REQ87, REQ88
runtime	Check the status of IaC deployment	Given the PIACERE IDE And An initiated IaC deployment When the user navigates to the IaC deployment record in the IDE And the user requests more details about the IaC deployment Then the IDE presents the user with IaC deployment status details	KR10, KR13	The user can check the status of IaC deployment.	REQ10, REQ12, REQ 55, REQ 81, REQ82, REQ83, REQ84, REQ85, REQ87, REQ88
runtime	Inspect PIACERE continuous performance monitoring	Given an initiated IaC deployment When the user navigates to the IaC deployment record in the IDE And the user requests to see Performance Monitoring Dashboard Then the user's browser is launched with the Performance Monitoring Dashboard shown And The user can navigate to the other Monitoring Dashboards	KR11, KR12	The user can request to see the Performance Monitoring Dashboard and navigate to the other related dashboards (Performance monitoring, Performance Self-Learning, Security Monitoring, Security Self-Learning dashboards).	REQ11, REQ16, REQ17, REQ46, REQ47, REQ50, REQ51, REQ52, REQ72, REQ92, REQ93, REQ94, REQ97

Phase	Title	Scenario	Main KR	Description	Requirements
runtime	PIACERE self-healing start	Given the deployment is defined in the IDE When the user runs the deployment in the IDE The user will be able to access the self healing log for that application in the IDE, this could be opened from the IDE in the Piacere runtime controller clicking the right button on the deployment There we will be able to see the related self healing actions taken, and we will be able to trigger manually some strategies.	KR12	The user can access the self healing log and can trigger manually some strategies suggested by the self-healing mechanism.	REQ16, REQ17

DRAFT

4 Integration Strategy (KR13)

4.1 Changes in v2

There are no changes respect version 1 on the strategy to follow for the continuous integration of the PIACERE solution. The sections 4.2, 4.3, 4.3 and 4.5 remain unchanged.

4.2 Integration strategy – definitions

The following terms and acronyms are used in this section.

Table 13: Terms and Acronyms for Integration Strategy

Terms used in section	Explanation of the term
High Availability (HA)	High level of availability of an IT system or application. This usually means that the system is installed in more than one instance.
Business Process Management (BPM)	A standard process for the management of business processes that is enabled through the use of Workflow / Process Engines.
Strategy	A general plan to achieve one or more long-term or overall goals under conditions of uncertainty.
Method	Detailed approach or solution to achieve a goal.
Integration strategy	Set of guidelines, assumptions and general directives related to the integration of components within a given IT system.
Integration	Alternative: process of linking together different components or systems in order to act as a coherent, coordinated whole.
Application Programming Interface (API)	The definition of the interfaces of a system or application made available to be invoked by external parties.
Enterprise Service Bus (ESB)	A method for integration of IT systems or components.
Enterprise Application Integration (EAI)	All tasks, activities, methods and tools used for integrating applications within an enterprise.
Representational State Transfer (REST)	A nowadays most common protocol for the integration of IT systems.
Message-oriented middleware (MOM) communication	Communication between IT systems based on a queue of messages, usually asynchronous.
Synchronous communication	Direct method of communication between IT systems, where the invoker is blocked until it receives a corresponding response.
Asynchronous communication	Indirect (usually through a queue message broker) method of communication between IT systems, where the invoker is not blocked until it receives the respective response.
Repository	A dedicated storage place where code and/or artifacts are versioned.
Branch	A movable reference to a commit that is interpreted as a sequence of such with the referenced commit being at the tip of the branch.
Tag	An unmovable reference to a commit, highlighting a certain commit for identification purposes, often meant to mean a certain state of the

Terms used in section	Explanation of the term
	repository, e.g., a particular version/revision of the software that was made available to the public.
Pipeline	A sequence of modules that facilitate a certain flow.
Flow	A sequence of actions that happens in a defined way.
Continuous Integration (CI)	The continuous process of integrating multiple software components to ensure they provide a coherent service.
Continuous Delivery (CD)	The continuous process of ensuring the latest integrated solution is available for installation (or already deployed).

4.3 Framework components

4.3.1 Integration Repository

The GitLab's CI/CD will be used for integration and testing. It needs certain configuration that will be provided by a central, integration, repository. The same repository will also host the descriptions of flows that are tested in that integration.

4.3.2 CI/CD Flow

The CI/CD flow will involve packaging the non-graphical components in containers and running example scenarios against the components as they run on Docker [5]. The flow will be largely based on the integration tooling as delivered in PIACERE Runtime Controller with the motivation described further below in the strategy section. The CI/CD flow will trigger on Pull/Merge requests to ensure that the code-to-be-integrated passes the defined tests.

4.4 Framework description DevOps Pipeline

The PIACERE framework components are version-controlled inside Tecnalia's GitLab using git [6] repositories branches and tags. Each component resides in a dedicated git repository as tracked by an internal spreadsheet. We plan to use GitLab's CI/CD functionalities to deliver the integration and testing pipeline. The interfaces offered by different components are described using OpenAPI and tracked in another internal spreadsheet as part of task 2.3 efforts.

GitLab was chosen as the already-available solution and its CI/CD were evaluated as matching the needs of the PIACERE project, and hence other solutions were not further evaluated. The features of CI/CD that were evaluated include:

- The ability to trigger on Pull/Merge requests.
- The ability to work across multiple projects/repositories.
- The ability to understand packaging and artifact distribution systems.
- The ability to integrate with code quality tools.

4.5 Selection of integration strategy

One main factor for the successful design and implementation of PIACERE is to provide a proper integration strategy that integrates the components on which PIACERE is built and thus mandates proper orchestration of the flow.

From the viewpoint of integration models, we investigate four popular integration strategies, including point-to-point integration, Message Oriented Middleware (MOM) integration, Enterprise Application Integration (EAI) or Enterprise Service Bus (ESB) based integration, and EAI/ESB integration with Business Process Management (BPM) orchestration.

The purpose of this section is to evaluate the different strategies for integration, and to select the most efficient according to the objectives of the PIACERE project. The selected strategy will also be analysed in order to highlight its main benefits and advantages.

The PIACERE framework integrates several underlying components into one platform. The proper selection of the integration architecture with PIACERE is a crucial point for the success of this project. An additional element to consider was the level of effort needed to implement the chosen integration method. A Business Process Management (BPM) orchestration was chosen as the most flexible and easy method of integration. BPMN (BPM Notation) [7] is a standard for the description and execution of business processes.

The key benefits of this approach are:

- Flexible logic implementation in the BPM flow with no hard coding.
- Support for both synchronous and asynchronous communication.
- Support for most of the integration protocols.
- Reliability, configuration easiness, and high availability.

For the BPM engine implementation, there are four possible solutions that have been evaluated:

1. Activiti [8] – one of the oldest and most mature open-source BPM implementations.
2. jBPM [9] – also, a mature and stable BPM implementation, developed by JBoss, with integration support for the business rule server Drools.
3. Camunda [10] – a mature and robust implementation of BPM, which does not require the whole JBoss stack to work.
4. Flowable [11] – the newest solution, developed by a team of former Activiti developers.

Based on our research and experience in other projects, Camunda has been chosen as the BPM implementation for the PIACERE project as it matches our requirements. The jBPM from JBoss requires the whole stack of the JBoss technology, which complicates the implementation of the project and increases the resource footprint of the platform. Key advantages of choosing Camunda are as follows:

- Lightweight implementation which is easy to deploy and maintain.
- Full support for the REST communication protocol.
- Easily available docker images, which allow for fast deployment.
- Low level of dependencies to other projects, which allows for easier upgrades and maintainability in the future.

Table 14: Integration Strategy Evaluation Criteria

Criteria	Activity	jBPM	Camunda	Flowable
Easy maintenance and deployment	Yes	No	Yes	Yes
REST support	Yes	Yes	Yes	Yes
Docker images availability	Yes	Yes	Yes	Yes
Easy upgrade and maintainability	No	No	Yes	No

5 Conclusions

The document described the updates realized in the second year of the project, also including parts that have remained unchanged to maintain consistency and completeness. It serves as an architectural document for the other work packages that are involved in developing the KRs of the PIACERE solution.

The list of requirements was updated with the new requirements presented in year two and it was completely revised thanks to the analysis process that allowed to accept the indications provided by the UCs and to consolidate the relationship between UCs and KRs.

The general architecture described in the document has been updated considering the indications derived from the development operations of the KRs carried out in second year. The revision of the internal functioning of the PIACERE components required a revision of the sequence diagrams, that illustrate their updated functionality, as well of the general workflows.

The document presented an approach to define the strategy for implementing multi-user functionalities, considering the technical decisions made in collaboration by the partners. It could be connected to the exploitation work-package for future evolutions of the PIACERE framework.

The study of the relationships between UCs and KRs led to the identification of some usage scenarios presented in this document, that will be extended in the deliverables of the related KRs.

The document also confirmed the integration strategy proposed in the first year of the project to integrate the components on which PIACERE is built and thus mandates proper orchestration. The combination of PIACERE Key Results and related components supports the extended DevSecOps approach.

Although this document is the final version and no further iterations are planned, the continuous improvement of the functionalities of the KRs and the validation of the UCs will be able to identify new requirements and optimizations of the general architecture.

6 References

- [1] ISO/IEC/IEEE International, «Systems and software engineering—Vocabulary,» 2017.
- [2] International Institute of Business Analysis, MoSCoW Analysis (6.1.5.2)". A Guide to the Business Analysis Body of Knowledge (2 ed.), 2009.
- [3] «Amazon Web Services,» [Online]. Available: <https://aws.amazon.com/>.
- [4] «Gherkin Reference,» [Online]. Available: <https://cucumber.io/docs/gherkin/reference/>.
- [5] «Docker,» [Online]. Available: <https://www.docker.com/>.
- [6] «GIT,» [Online]. Available: <https://git-scm.com/>.
- [7] «Business Process Model and Notation,» [Online]. Available: <https://www.bpmn.org/>.
- [8] «Activiti,» [Online]. Available: <https://www.activiti.org/>.
- [9] «JBPM,» [Online]. Available: <https://www.jbpm.org/>.
- [10] «Camunda,» [Online]. Available: <https://camunda.com/>.
- [11] «Flowable,» [Online]. Available: <https://www.flowable.com/>.

DRAFT

APPENDIX: PIACERE Glossary

Changes in v2

The glossary has been updated including the IaC Scan Runner component that acts as KR6-KR7 executor. Although no further changes have been made, the entire glossary is reported in the appendix to maintain completeness.

Glossary structure

The Glossary is structured in two main sections. The first called Basic Terms defines the terms used for the PIACERE project. The second section indicates the components expected for the project and their descriptions. Below there is a logical diagram of how the second section is composed. The items indicated are indicative and not mandatory.

Functional Description: [Description of the components functions and features, what part of the PIACERE workflow is covered. This includes the standard workflow.]

Input: [What this component takes as input (models, JSON payload, blueprint or similar)]

Output: [What this component returns as output (file, entry or log in system, response)]

Programming languages/tools: [Python/Java/.NET/ ...]

Dependencies: [On other internal or external components with specific interaction description]

Critical factors: [Any critical factors that may include errors in the received inputs, configuration and mitigation.]

Basic Terms

The application

As PIACERE is considering the application components to be a black box, we must define the line between the application itself and the IaC. The aim is to have as clear division and understanding of what the application and IaC actually can be. The main actor is the user, which decides the granularity of the application and the corresponding IaC to be modelled in the PIACERE. We model IaC required to run the application and not modify the application components themselves. The configuration files, FRs, TRs should be provided in DOML to successfully model, deploy and manage the application. The aforementioned configuration files, FRs, TRs are part of the DOML.

NOTE: please see the DOML definition.

Technical Requirements (TR)

The explicit requirements concerning the infrastructural elements to be used for a certain application. These are provided by the end-user in charge of modelling the application deployment.

Under Technical Requirements we deem explicit requirements for:

- The characteristics of computational environments and networks – e.g., CPU, memory, cores

- The type of computational environments and networks – e.g., AWS S3 services, Kubernetes, Google Cloud, etc.

Non-Functional Requirements (NFR)

The explicit requirements, provided by the end-user modelling the application deployment, concerning the non-functional properties of the application that will be running on top of the infrastructure.

Under Non-Functional Requirements we deem explicit requirements for the response times, availability of the infrastructure, cost, etc.

Note that in PIACERE we do not focus on the functional requirements offered by a certain application and, in fact, the PIACERE platform is completely agnostic with respect to this aspect.

Configuration Management

Configuration Management: by infrastructure configuration we mean the process that enables to create and update a software environment on existing servers according to a given set of requirements. This means for example installing software packages, then configuring and starting them, but also configuring networks.

e.g., Chef, Puppet, SaltStack, xOpera, Ansible, CFEngine.

Infrastructure Provisioning

Infrastructure Provisioning: help in automating the basic lifecycle steps of infrastructure resources: create, update, and delete. These provisioning steps usually target virtual resources, either on premises or in the cloud, such as Virtual Machines (VMs), but can also target physical resources by using suitably flexible hardware platforms such as HPE Synergy.

e.g., Terraform, AWS CloudFormation, xOpera, OpenStack Heat.

Orchestration

Orchestration: it is a process composed of a set of workflows of low-level operations like provisioning of resources, configuring and installing components, connecting components to apply dependencies, or tear down individual components. Orchestrators can work with any of the resource types – compute, networking, storage, services and more.

e.g., Apache Brooklyn, Alien4Cloud, xOpera, Cloudify, ARIA TOSCA, OpenTOSCA, Kubernetes, OpenStack Tacker.

Container Orchestration

Container Orchestration: It is the set of processes to automate the deployment, runtime management, scaling, and networking of containers. Examples of tools that support these processes are Kubernetes, Docker Swarm.

Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is the code needed to automate provisioning of resources, their configuration, the deployment of software components on top of them, their configuration and execution. The initial set of IaC languages, as described in DoA, is Terraform, TOSCA and Ansible.

This automation eliminates the need for developers to manually provision and manage servers, operating systems, database connections, storage, and other infrastructure elements and application components.

It promotes managing knowledge and experience of plethora of subsystems as a single commonly available source of truth instead of traditionally reserving it for system administrators.

Infrastructure as a Service (IaaS)

A **platform** is described as a collection of hardware and software components that are needed for a software tool used for computer-aided software engineering (CASE) to operate. As cloud computing has grown in popularity, several different models and deployment strategies have emerged to help meet specific needs of different users. Each type of cloud service and deployment method provides with different levels of control, flexibility, and management.

Among **Cloud Computing Models, Infrastructure as a Service (IaaS)** contains the basic building blocks for Cloud Information Technology, and typically provides access to networking features, data storage space, and computing nodes (either virtualized or running on dedicated hardware).

Typically, in IT industry, the fewer the abstraction layers, the more control one has over resources, and the lower the payments to mediating service providers. This works both ways, as a lower abstraction level involves higher complexity, but lower costs if one is capable to control efficiently and effectively all related intricacies.

More details in IaaS and other Cloud Computing Models can be found in the addenda.

Target IaC Language (TlaCL)

DOML models define the organization of software applications in terms of components and connectors and their mapping into middleware level and infrastructural components. Such models must be translated into executable Infrastructure as Code formats that can be used to automate the phases concerning provisioning and configuration of the infrastructure and the deployment, configuration and operation of middleware and application-level components.

A target IaC language is one of the executable IaC formats into which PIACERE can translate DOML models. PIACERE will offer translators for at least Terraform for provisioning of infrastructural elements and Ansible for the other configuration and deployment steps. Other IaC target languages could be plugged into the platform by exploiting the PIACERE extension mechanism.

Configuration Drift

In this project we can consider two levels of configuration drift:

- configuration drift happens when, usually due to manual intervention, the hardware and software infrastructure configurations “drift” or become different in some way from the IaC that generated the configuration.

It is possible to call Configuration Drift also the modification of IaC with respect to DOML that generated it:

- any changes to the IaC, deployed application or the runtime infrastructure not stemming from PIACERE (i.e., DOML or any PIACERE component) is considered a configuration drift and as such, undesired state. Please see the definition of DOML.

DevOps Modelling Language (DOML)

The DevOps Modelling Language (DOML) is the language PIACERE offers to its end-users (DevOps team members) to allow them to describe the external structure of their application (seen in terms of black-box components to be deployed) together with any technical and non-functional requirement concerning the infrastructure to be provisioned and configured to run such an application.

The DOML allows PIACERE end-users to work at different levels of abstraction and, thus, to incrementally specify a set of sub-models that include the following elements:

- The application structure using the modelling abstractions that are made available at the Application Layer.
- The underlying abstract resources to be used and their association to the application's components. In this step the abstractions made available at the Abstract Infrastructure Layer are used.
- Finally, the concretization of the previous model in terms of concrete resources offered by concrete providers. This is done by relying on the abstractions made available at the Concrete Infrastructure Layer.

We separate the Abstract Infrastructure Layer from the Concrete Infrastructure Layer to allow users to produce models that can have multiple realizations. This allows, on the one side, to have people with different roles and competences intervening at the different layers. On the other side, it offers a tool to easily change concrete resources, while keeping models at the higher levels unaltered.

The information inserted in the models at the various levels will allow provisioning, configuration, deployment and runtime orchestration activities to be executed. More specifically, the Concrete Infrastructure Layer will be used to generate IaC for provisioning purposes. The other layers will provide information relevant to the generation of the IaC relevant for the other purposes.

Infrastructure Element (IE)

A single entity that is both modelled in DOML and later managed in PIACERE runtime.

PIACERE design time

PIACERE design time is the (time) scope of the PIACERE project that involves the initial tasks to design the desired infrastructure using the PIACERE tooling as well as any further user-driven process involving modifications in the initial design.

PIACERE design time involves such components as: IDE, DOML, ICG, VT.

PIACERE runtime

PIACERE runtime is the (time) scope of the PIACERE project that involves managing the running infrastructure that was previously designed at design time.

PIACERE runtime involves some shared components from the design time as helpers (ICG, VT) and means of communication (DOML).

PIACERE runtime operates using one or more Target Environments

PIACERE runtime is responsible for implementing and managing the Execution Environment. PIACERE runtime is mainly comprised of the following components: PIACERE Runtime Controller (PRC), IaC Executor Manager (IEM), Infrastructure Advisor (IA).

Resource Provider (RP)

PIACERE is creating/using resources through the selected Target IaC Language and tooling on Resource Providers, to create the Execution Environment for the application.

Examples of: AWS and friends, OpenStack, bare-metal, IoT

NOTE: This (as well as TlaCL) was mentioned as Target Environment in the DoA.

Cloud Service Provider (CSP)

One kind of cloud resources provider, e.g., Amazon's AWS, Google's GCP, Microsoft's Azure, Alibaba Cloud, some OpenStack.

Production Resource Provider (PRP)

The production (non-canary) variant of the Resource Provider (RP).

Canary Resource Provider (CRP)

The canary (non-production) variant of the Resource Provider (RP).

RPs of this kind are provided by the Canary Sandbox Environment (CSE) task.

They come in two variants: real and simulated, i.e., with mock-ups.

Mock-up

A functionality, which has the same API as an existing infrastructure provider (e.g., AWS) and returns the success/failure along with the expected data that would be returned from the real API call.

It is used in the simulated variants of the Canary Resource Provider.

For more details see the Canary Sandbox Environment in Components.

Execution Environment (EE)

The Execution Environment is essentially what we model in DOML and then realise through IaC, up to the point when we deploy the application and run it. **The Execution Environment is thus an environment in which the application is running.** It can span over different CSPs, different technologies (i.e., may be heterogeneous). Any non-user changes of the Execution Environment are realised through the Optimizer (IOP), either in the initial phase or when invoked by the SelfHealing component. All non-user changes are reflected in the updated DOML. User changes are considered a Configuration Drift.

PIACERE runtime creates the EE using the DOML converted to IaC and run using appropriate tooling.

Production Execution Environment (PEE)

Production Execution Environment (PEE), in the strict sense, is an EE that is hosting the application on an infrastructure, built using DOML and implemented by IaC, for production purposes.

In the weaker sense, it is any EE that is not a Canary EE.

Canary Execution Environment (CEE)

Canary Execution Environment (CEE) is an EE that is created using one or more Canary Resource Providers. It might or might not allow to run any steps beyond the infrastructure deployment, e.g., it might be entirely mocked up and not use any resources it claims to have.

Components

Integrated Development Environment (IDE)

Functional Description: The PIACERE IDE (Integrated Development Environment) will be a tool for modelling and verifying IaC solutions following the Model-Driven Engineering (MDE) approach. The IDE will enable to define IaC at an abstract level independently of the target environment based on the PIACERE DOML (DevOps Modelling Language).

Input: No inputs

Output: A DOML instance of the solution to be deployed.

Programming languages/tools: Eclipse Theia + EMF Cloud

Dependencies: The IDE will integrate the Verification Tool (VT) and Infrastructural Code Generator (ICG). Thanks to the VT, it will be possible to validate the defined models and to make suggestions, possible substitutions and improvements. Through the ICG tool, the corresponding IaC in a specific target environment (e.g., Terraform, Ansible, TOSCA...) will be automatically obtained.

Critical factors: The IDE will be designed to be extensible, so to allow the new IaC tools and the new abstractions of infrastructural components that will be incorporated into DOML (DOML-Extensions).

Infrastructural Code Generator (ICG)

Functional Description: This component generates the required IaC from DOML and possibly, the configuration files. The proposed DoA IaC languages are Terraform and Ansible with possible extensions to Chef, Puppet, SaltStack. The conversion from DOML into IaC is a pure function¹ that is, deterministic. ICG may generate IaC for different tools/languages, according to the DevOps activity to be automated (Provisioning, Configuration, Deployment, Orchestration). ICG will be a command-line tool, reading input from and writing output to the underlying file system, like common compilers do.

Input: File from DOML (the files could be more than one).

Output: File containing code in the chosen target IaC language (the files could be more than one, possibly organized in a directory structure as defined by the respective target tool).

Programming languages/tools: Python

Dependencies: ICG has dependencies on the DOML source and the target service provider.

¹ In computer programming, a pure function is a function that has the following properties:

1. The function return values are identical for identical arguments
2. The function application has no side effects

https://en.wikipedia.org/wiki/Pure_function

Critical factors: ICG needs to know the target provider because the infrastructure component definitions (in Terraform) are provider-specific.

Canary Sandbox Environment (CSE)

CSE is one of key results within PIACERE. The goal is to provide tools that would allow to dynamically test the IaC in a fast and cheap manner. The tools are described in the following subsections: CSEP and CSEM. There are two approaches to the CSE: to provide a real (non-simulated) Canary Resource Provider and a simulated one. Depending on the variant, the scope and characteristics of testing differs. Real providers require resources and allow to complete all steps of deployment as long as the supporting infrastructure (beneath the created provider) is sufficient. The assumption is that the user is able to provide the hardware (e.g., because they have bare metal or virtual machines, either on premise or elsewhere – the CSE is agnostic to that). On the other hand, the simulated variant does not consume resources but does not allow further steps other than provisioning of the infrastructure elements.

Note: CSE can be used to test other relevant PIACERE components, e.g., IEM.

Properties possible to be studied using a Canary Resource Provider are:

- Technical Requirements (TR)
 - Are the right resources really provided?
- Security (security testing) – e.g., if connections are allowed or not
 - Limited to infrastructure elements in the simulated case
 - Allows DAST in the real case
- Robustness (stress testing) – e.g., if the VM creation fails, how to react.
 - Limited in the simulated case – it might be too permissive due to no real constraints
- Integration test or “Completeness”, that is check if everything is deployed correctly, every connection is properly opened, every component is properly connected, etc.:
 - Are all network segments defined?
 - Do we have connectivity from VMs (internal/external)?
 - Only a declaration-based check in the case of simulation
- In the real case also configuration tests via tools like Serverspec

Examples of properties NOT possible to be studied within the CSE AT ALL are:

- Non-Functional Requirements (NFRs)
 - The performance
 - It will either differ from the production (in the case of a real provider) so not useful or not be measurable at all (in the case of a fake one).
 - All others are not applicable at all as there is no notion of cost, availability, region, policies etc.

Canary Sandbox Environment Provisioner (CSEP)

Functional Description: The role of this component is to create the desired Canary Resource Provider(s). This may entail provisioning and configuring new systems to provide the desired platform. The initial set of supported providers is OpenStack (for real [non-simulated] actions) and CSEM (for simulation, see below). The discussion continues on whether we consider Docker Swarm and/or Kubernetes at this level. Note: they might be deployed further on top of OpenStack for flexibility.

Note: An interesting case would be to actually use PIACERE toolset to be the basis for CSEP but it is a chicken and egg problem at the moment.

Input: The input to this component constitutes the configuration with respect to what Canary Resource Provider(s) should be provided and what their config values are.

Output: This component returns information on the provisioned Canary Resource Providers including but not limited to: API endpoints, credentials.

Programming languages/tools: Python

Dependencies: ICG must be able to generate code compatible with deployable Canary Resource Providers. Weak dependency on CSEM (CSEP needs to know how to deploy it). Other PIACERE components may depend on it to provide a testing environment for PIACERE itself.

Canary Sandbox Environment Mocklord (CSEM)

Functional Description: The role of this component is to simulate an existing resource provider so that the user can easily test interactions against it. The plan is to research the usefulness of such approach to dynamic IaC testing. The prototype will target a subset of AWS APIs. CSEM is deployed and configured by CSEP and is assumed to have much lower cost compared to real (non-simulated) resource providers. Due to simulation, this variant of Canary Resource Provider will allow only the provisioning step to happen.

Note: it is unlikely to be able to guarantee 100% compatibility with the mocked provider (e.g., AWS) due to them being effectively black boxes.

Input: It should allow API calls allowed by the provider being mocked.

Output: This component records the state of the mocked-up environment and allows to retrieve information on it, e.g., created VMs, opened ports.

Programming languages/tools: Python + e.g., moto library for mocking AWS

Dependencies: ICG must be able to output IaC compatible with the simulation (i.e., the provisioning step must be separate from further ones). Infrastructural Services Catalogue might be used to decide on offered resources dynamically (e.g., types of VMs) - note: this should be the same functionality as the one required by IOP already – to know “the offer” but it can also be configured via a side channel. Other PIACERE components may depend on it to provide a testing environment for PIACERE itself.

DOML & IaC Repository

The DOML models, as well as the generated IaC, will be stored in the user’s file system or, upon a proper configuration of the IDE, in a version management system such as git. This will give the possibility to all PIACERE component to share the DOML model files by using the corresponding links. This will also allow multiple versions of a DOML model to be available and used by different tools if this will be necessary.

Infrastructural Elements Catalogue (IEC)

The Infrastructural Elements Catalogue is a required service for the optimizer (IOP) and it contains the description (NFR, TR and dynamic runtime metrics) of the available IEs to be considered in the optimization process by the IOP

Each item within the Infrastructural Elements Catalogue is associated with the historical data on the important properties of the infrastructure, emanating from the monitoring data:

- Real availability
- Real response times,
- Etc.

This information (dynamic monitored data) along with the static characteristics of the infrastructural elements will serve for the IOP to select the best combination of infrastructural elements given a set of TRs.

Initially the catalogue will include basic infrastructural elements (VMs + storage + IoT gateways) and then it will be enlarged with other types of elements such as Kubernetes.

Verification Tool (VT)

The VT focuses on static analysis of the IaC (IaC Static Verification).

The VT consists of the following components:

- **Model Checker:** Given a DOML description checks for the consistency and completeness of the DOML and associated topology. It would be possible to provide some correctness properties given in a suitable DOML sub-language. The VT provides the outputs:
 - Yes, the provided DOML is consistent and complete.
 - No, the DOML should be changed – provides suggestions on what are the problems and (possibly) ways to fix them.
 - (Correctness): Yes, the provided DOML satisfies the correctness properties.
 - (Correctness): No, the provided DOML is not correct and at least one counter-example is provided.
- **IaC Static and Security Verification**
 - **BASIC:** Yes: correct & complete; No: provides suggestions on what is to be changed.
 - **ADVANCED:** to evaluate the IaC code for quality, maintainability – check SonarCloud (currently does not support IaC).
- **Security Components Inspector:** provides checks of the cryptographic libraries to be used within the application deployment using the DOML, IaC and configuration files provided.

Model Checker

Functional Description: The Model Checker performs the following checks, based on DOML:

- Checking whether the model is consistent and complete (e.g., there are no dangling connections, all components have defined a corresponding infrastructure...).
- Checking whether data flow from a component to the other according to the defined constraints (e.g., for privacy reason, certain pieces of data cannot reach some component A).
- Checking whether the model complies with the properties provided by the user, if present.

Input: DOML model

Output: Yes/No and a counterexample in case of a negative result

Programming languages/tools: Python, Z3 SMT solver

Dependencies: IDE – the IDE will provide the input and consume the output.

Critical factors: DOML syntax compatibility

laC Security Inspector

Functional Description: The laC Code Security Inspection provides the laC static tests - SAST tests, using the tools from the open-source communities. The laC is tested against predefined policies (TR, NFR), enabling regulation of the laC code based on the overall company policies and against the potentially harmful laC code patterns.

The component will follow these steps:

- Traverse through laC, find a set of dependent/used libraries in laC
- Check versions (detection of vulnerable ones)
- Check configuration (i.e., ports, credentials)
- Check whether inputs are valid
- Find hardcoded usernames/passwords, etc. and typos
- License check
- Prepare output (warn, recommend).

Input: API or CLI call takes as input the laC code, generated by the ICG.

Output: A set of warnings and recommendations as a response to the API call.

Programming languages/tools: Python

Dependencies:

- ICG – the Infrastructural Code Generator will provide the input.
- IDE – the IDE will consume output from the component.

Critical factors: Any critical factors that may include errors in the received inputs, configuration and mitigation.

Component Security Inspector

Functional Description: An analyser and ranker of components (libraries, middleware) from a security point of view. Code Security Inspector will extract dependency information from the laC, detect included programs and libraries with known vulnerabilities by querying public vulnerability databases in order to produce a report to the PIACERE user (IDE), informing the user about the appropriateness of the components included in their solution.

Main functionalities:

- Cryptographic software libraries will be analysed
- Most appropriate frequently used (based on used modules within laC) cryptographic libraries will be selected
- The tool will include tests for attacks against them
- This tool will verify vulnerabilities by using carefully designed test cases to execute libraries' functions and observe their behaviour and output to detect the possibility of attacks.
- Tests will be made periodically.

Process and steps of the tool: prepare knowledge base of crypto libraries, check if libraries are used (subset of SAST libraries), check versions/configuration, prepare output (warn, recommend).

Input: laC code, generated by the ICG

Output: A set of warnings and recommendations

Programming languages/tools: Python, Java

laC Scan Runner

Functional Description: A component combining laC Security Inspector (KR6) and Component security inspector (KR7). Apart from features of these two distinct parts, laC Scan Runner also introduces improved scan result aggregation and summary, in form of visual tabular representation, ordered with respect to outcome results within HTML web page. Additionally, it also includes scan result persistence and re-use of user-defined preferences in form of configurations. It requires MongoDB document store in order to leverage the persistence layer

Input: laC code, generated by the ICG

Output: A set of warnings and recommendations with visual summary (HTML page) showing sorted results with respect to their outcomes.

Programming languages/tools: Python, MongoDB

Dependencies:

- ICG – the Infrastructural Code Generator provides the input.
- IDE – the IDE consumes output from the component.

Critical factors: Any critical factors that may include errors in the received inputs, configuration and mitigation.

Dependencies:

- ICG – the Infrastructural Code Generator provides the input.
- IDE – the IDE consumes output from the component.

Critical factors: Any critical factors that may include errors in the received inputs, configuration and mitigation.

PIACERE Runtime Controller (PRC)

Functional Description: This component is the main control component of PIACERE runtime. It is a state machine that guides the overall workflow within PIACERE runtime. Actions of PRC are targeted against a specified set of resource providers (including Canary and Production).

Input: This component receives messages of two types: events (notifications) and commands (RPCs) from other components via a queue interface.

Output: This component produces further messages which are placed in the queue system and handled by other components.

Programming languages/tools: Java + Camunda BPM + ActiveMQ

Dependencies: This component does not strictly depend on other PIACERE components, but it interacts with other PIACERE components, mostly runtime: including laC Executor Manager (IEM), which it controls, and Infrastructure Advisor (IA) which it sets up and communicates with (note: IA is made of several distinct components). Absence of these means there is no real work being done by PRC. Similarly, IDE interacts with PRC.

Critical factors: The received messages may be mis-formatted and hence un-handable. Sent messages may have no receivers or receivers are unable to handle them. The queue system might fail.

Comments/open questions/issues: Who/what sets up PRC? Also, I see some components have already declared to be offering REST APIs – are we coupling the services using API endpoints then? Would not a queue be a better fit here? At least for the runtime components.

IaC Executor Manager (IEM)

Functional Description: its purpose is to plan, prepare, and provision the infrastructure and the corresponding software elements needed in the deployment. This work entails the following activities: i) creation of the underlying infrastructure, ii) sort out the software dependencies and configuration, iii) deployment of the applications, iii) un-deploying applications/cleaning.

Input: API or CLI call takes as input the IaC code, generated by the ICG.

Output: a code stating the deployment status.

Programming Languages/Tools: Python, IaC Tools.

Dependencies:

- ICG – the Infrastructural Code Generator will provide the input through the PIACERE Runtime Controller

Critical Factors:

- The received IaC scripts may contain errors.
- Connectivity issues with the different components (e.g., Cloud providers, devices, Container Orchestrators).
- Security concerns during the communication.
- Authentication and authorization issues during the deployment.

Infrastructure Advisor (IA)

Infrastructure Advisor holds four main sub-components:

IaC Optimizer Platform (IOP)

Functional Description: The optimization problem formulated in PIACERE and solved by the IOP consists on having a service to be deployed and a catalogue of infrastructural elements, with the principal challenge of finding an optimized deployment configuration of the IaC on the appropriate infrastructural elements that best meet the predefined constraints (e.g., types of infrastructural elements, NFRs, and so on). In this context, it is the IOP component which is the responsible for finding the best possible infrastructure given the input data received. This input data is provided in DOML format and will include the optimization objectives (such as the cost, performance, or availability), optimization requirements and previous deployments (in case it is necessary). Then, the IOP performs the matchmaking for the infrastructure by the execution of optimization intelligent techniques using the information taken as input against the available infrastructure and historical data, available from the catalogue of Infrastructural elements

Input: The input of the IOP can be divided into two aspects:

- DOML (which consists of the FR, TR, The infrastructure model (i.e., VMs, K8S, etc), the configuration (e.g., application specific YAML, Docker, etc. definitions))
- Information (static + dynamic) from the Infrastructural elements catalogue.

Output: IOP will provide its result (the selected optimized infrastructural elements) in DOML (PSM level).

Programming Languages/Tools: Java.

Dependencies: Run time monitoring system. This component has access to DOML.

Critical factors:

- The IOP must be “fast” – the IOP will search through a potentially large solution space – the complexity of the NFR/TR influences the choice of optimisation algorithm.
- The IOP should work on two different scenarios: first deployment, and as result of an action raised by the SelfHealing. In the first of the cases, the IOP should return several solutions optimizing all the objectives considered. In the second case, the IOP should return a working solution in a fast time, which amends the problem detected.
- The optimization problem to solve is a multi-objective one.

Monitoring Controller

Functional Description: This component concentrates the infrastructure resource monitoring activation and deactivation activities throughout all the monitoring components: performance monitoring, security monitoring, PerformanceSelfLearning and SecuritySelfLearning.

Input: Data provided by the PIACERE Runtime Controller, specifically the id of the application from which we must monitor their resources.

Output: An acknowledge that the request has been received and it is being processed towards the monitoring and SelfLearning components.

Programming languages/tools: Python

Dependencies: PIACERE Runtime Controller.

Critical factors:

- We require that the monitoring agents label their metrics with the application id.
- The usage of the application id label may constrain the usage of the same infrastructure resource to provide or support components from different applications.

Open questions:

- How to manage the situation of several applications running in the same infrastructure resource.

Monitoring

Under monitoring we currently cover two non-functional aspects: performance and security.

Performance Monitoring

Functional Description: This component concentrates the infrastructure resource monitoring activation and deactivation activities throughout all the monitoring components: performance monitoring, security monitoring, PerformanceSelfLearning and SecuritySelfLearning.

Input: Data provided by the PIACERE Runtime Controller, specifically the id of the application from which we must monitor their resources.

Output: An acknowledge that the request has been received and it is being processed towards the monitoring and SelfLearning components.

Programming languages/tools: Python

Dependencies: PIACERE Runtime Controller.

Critical factors:

- We require that the monitoring agents label their metrics with the application id.
- The usage of the application id label may constrain the usage of the same infrastructure resource to provide or support components from different applications.

Open questions:

- How to manage the situation of several applications running in the same infrastructure resource.

Security monitoring

Functional Description: The Security monitoring system consists of subsystems (Wazuh deployment – manager and agents - with specific components for data transformation) collecting data in order to provide values for security metrics. As an additional option it can provide the deployment of Vulnerability Assessment Tool (VAT) that is capable of monitoring API end-points of the specific Web Application.

Input: Metrics defined by the NFRs and TRs from the DOML. Additional to the NFR and TR monitoring, we are monitoring security metrics: e.g., Security of the configuration – metrics are not defined right now – but could be the check of the component versions; mapping between CVEs and components; configuration changes, not prescribed by the IaC – potential action to enforce redeployment.

Output:

- The classified events are sent to SelfHealing component to be further inspected.
- The data collected is used by SecuritySelfLearning component to analyse/classify events (detect anomalies)

Programming languages/tools:

- Wazuh, VAT: Python, C++, JavaScript

Dependencies:

- Wazuh deployment, Ansible
- Vulnerability Assessment Tool deployment (VAT)

Critical factors:

- “The price” for running complete monitoring stack might be of high impact
- Configuration of the deployment of Wazuh and the Vulnerability Assessment Tool

Open questions:

- Dynamic configuration step of the monitoring components.

Self-Learning

Under monitoring we currently cover two non-functional aspects: performance and security.

PerformanceSelfLearning

Functional Description: This component predicts malfunctioning (TRs degradation) and detects the *concept drift phenomenon* and/or *anomalies* in data provided by the Runtime monitoring system, and then it warns the SelfHealing component to be triggered. Any event threatening the QoS of an IaC deployment should be detected. Therefore, this component might have two

different modules: one module to detect the *concept drift phenomenon* and another one to detect anomalies.

Input: Data provided by the Runtime monitoring system, which may suffer from *concept drift* and/or *anomalies*.

Output: A response for the SelfHealing component, which may be an alert of the potential failure in one/several considered variables, e.g., infrastructural element, potential failure (which TR, even the metric), etc.

Programming languages/tools: Python

Dependencies: Run time monitoring system. This component has access to DOML.

Critical factors:

- Is this component trained in a real-time mode or with historical data every concrete period of time? Or even is it trained only once with historical data at the beginning of the IaC life? According to DoA: "... The self-learning mechanisms will manage their own training phase based on historical information from the runtime infrastructure (i.e., past failures) ...", but in other sentences DoA uses the terms "real-time", "incremental learning" and "run time". We must deal with this issue at this stage of the project. From my perspective, a real-time learning makes more sense.
- Data provided by the Run time monitoring system has to show evidences of *concept drift* or *anomalies*, otherwise this component wouldn't make sense, and therefore the SelfHealing component wouldn't be triggered.
 - We are currently unsure on the type of data but can assume it is time-series (TS) data, that indicates the status of the platform. In case of being TS, the streaming and the concept drift approach should address the temporal dependence issue.

Open questions:

- Not sure how the data will look like (time-series/ status/ version number), even the characteristics of attributes (how many, types, meaning of each attribute, will they be enough for our detection purposes?)
- Expected state of the infrastructural elements compared to the actual state (GT is DOML)

SecuritySelfLearning

Functional Description: The SecuritySelfLearning component receives data from the SecurityMonitoring component. As a first necessary step, a specified subset of the data has to be used to train a behavioural model. This subset of data, along with the necessary configuration files, is provided to the ModelTraining component, which eventually stores every trained model in the ModelRepository. Once a model is trained, this step is repeated only if requested to do so. A trained model is loaded from the ModelRepository to carry out anomaly detection of the data received from the SecurityMonitoring component. Under previously specified conditions, e.g., high number of anomalies in a short time period, the SecuritySelfLearning component will notify the SelfHealing component.

Input:

Data stemming from the Security Monitoring component.

Programming languages/tools:

Python

Dependencies:

- Grafana dashboard (deployment).

Critical factors:

- Building the model for the anomaly detection.

Open questions:

- The process of building the model is still open – it needs to be run either in parallel on a different deployment of the application or needs to be already built beforehand if it is used for the anomaly detection.

Self-Healing

Functional Description: The Self-Healing component gets input from the Monitoring and SelfLearning components both performance and security and will assess what should be changed within the infrastructural elements (if needed), to correct the (potential or actual) error or failure. It receives the input, classifies the event and launches the corresponding mitigation actions.

Based on the type of alert received from the monitoring components SelfHealing strategies will be sent to the PIACERE runtime controller that will perform some actions that will have to be identified as part of the strategy. Examples are:

- Launch the IOP
- Reboot machines
- Scale up the infrastructure
- Trigger the orchestration execution through the runtime orchestrator

Input: It will be launched by the SelfLearning or the runtime/security monitoring and as input it will receive information about the event originating the failure.

Output: As output it will generate a set of actions to be performed (call the IOP, etc) by the orchestrator.

Programming languages/tools: Java

Dependencies: Monitoring components: PerformanceMonitoring, SecurityMonitoring, PerformanceSelfHealing and SecuritySelfHealing.

Open Questions:

- We need to understand what we can request the PIACERE runtime controller (PRC), as the strategies in principle are going to be workflows that we intend for the PRC to run. However, there are some aspects such as the required information that we should check per each strategy.

Addenda

This section includes expanded information on some of the topics

IaaS and Cloud Computing Models

IaaS provides the highest level of flexibility and management control over IT resources, in contrast with the **Platform as a Service** Cloud Computing Model (**PaaS**), which removes the need for an organization to manages the underlying infrastructure. Therefore, IaaS is a Managed

Infrastructure C. C. model, which provides surgical configuration control over infrastructural resources, while removing an abstraction layer.

Some examples of tools used in PaaS models are Terraform (an open-source infrastructure as code software tool that provides a consistent CLI workflow to manage hundreds of cloud services, by codifying cloud APIs into declarative configuration files), and Docker (which uses Operating System-level virtualization to deliver software in packages called containers), although both of these tools also include IaaS features.

The third Cloud Computing model: **Software as a Service (SaaS)** is designed with the highest level of abstraction as seen by the end user, since the Platforms management tasks are also abstracted and supplied by a SaaS vendor. A common example of a SaaS application is web-based email.

Schematically, as ordered by decreasing abstraction level, and increasing control over resources:

SaaS >> PaaS >> IaaS

Consequently, IaaS models interact intensively with **Infrastructure as Code (IaC)**, commonly described within templates. These templates do detail all aspects of the underlying infrastructural elements that are to be managed, an activity which may involve tasks such as deployment, configuration, and release/deallocation of resources.

Since Infrastructure as Code (IaC) is the practice of managing infrastructure in a file or files, rather than manually configuring it via a user interface, infrastructure resource types managed with IaC can include virtual machines, security groups, network interfaces, and many others.

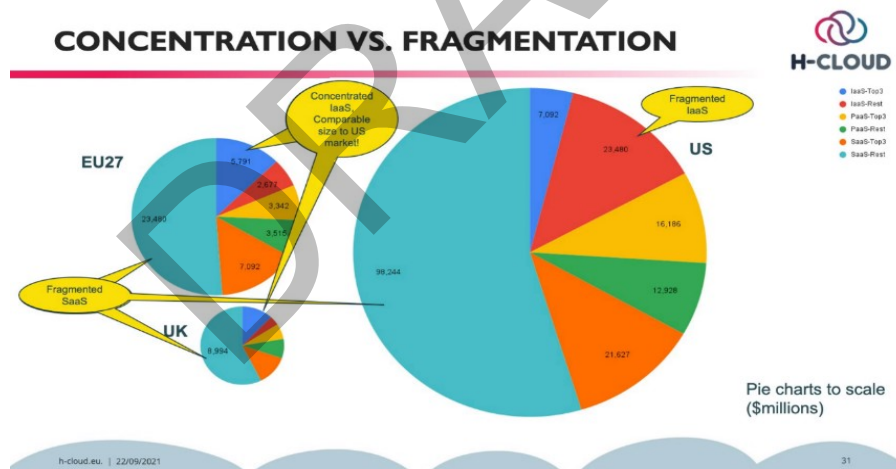


Figure 29: Status of cloud computing models (source: H-cloud)

According to H-Cloud’s presentation of the consultations held for the Strategic Report on Cloud Adoption (<https://www.h-cloud.eu/>), SaaS is by far the most popularly adopted Cloud Computing model among respondents from the EU, the UK, and the USA (v. chart), though in the EU, and in the UK, IaaS models are significantly larger than they are in the USA (which is by far the largest market in volume as a percentage of GDP).

Consequently, there seems to exist an opportunity for Europe to leverage their proportionally higher IaaS Cloud Computing Models adoption rate, on international markets. But also, to increase their proportional adoption rates across all of the cloud servicing spectrum.