Deliverable D2.5

PIACERE DevSecOps Framework – v3

Editor(s):	Radosław Piliszek (7bulls)
Responsible Partner:	7bulls
Status-Version:	Final – v1.0
Date:	31.08.2023
Distribution level (CO, PU):	Public

Project Number:	101000162
Project Title:	PIACERE

Title of Deliverable:	PIACERE DevSecOps Framework – v3
Due Date of Delivery to the EC	31.08.2023

Workpackage responsible for the Deliverable:	WP2 - PIACERE Requirements, Architecture and DevSecOps
Editor(s):	Radosław Piliszek (7bulls)
Contributor(s):	Radosław Piliszek (7bulls), Ismael Torres Boigues, Eliseo Villanueva Morte (Prodevelop), Gorka Benguria, Josu Díaz de Arcaya, Eneko Osaba, Iñaki Errazkin (Tecnalia), Laurentiu Niculut, Debora Benedetto (HPE), Grega Redek, Aleš Černivec, Matija Cankar (XLAB), Michele Chiari (POLIMI)
Reviewer(s):	Juncal Alonso (Tecnalia)
Approved by:	All Partners
Recommended/mandatory readers:	Mandatory: WP2, WP3, WP6 Recommended: WP4, WP5

Abstract:	This deliverable (D2.5) is v3, the final one, of the series of deliverables describing the PIACERE DevSecOps Framework, its integration and the information about the runtime-gluing component – the PIACERE Runtime Controller (PRC). Different versions of the PIACERE DevSecOps Framework have been provided following an incremental approach. The first version was an initial prototype with the core functionalities implemented; the second version augmented these functionalities taking into consideration the feedback coming from the use cases analysis; this final version includes final corrections and feedback coming from the implementation of the use cases.
Keyword List:	Framework, integration
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	29.06.2023	First draft version	Radosław Piliszek (7bulls)
v0.2	28.07.2023	Integration and PRC information updates Final version for internal review	Radosław Piliszek (7bulls) and contributors
v0.3	21.08.2023	Updates post internal review	Radosław Piliszek (7bulls), Juncal Alonso (Tecnalia) and Matija Cankar (XLAB)
v1.0	31.08.2023	Final check. Ready for submission	Juncal Alonso (Tecnalia)

Table of contents

List of tables

List of figures

Terms and abbreviations

API	Application Programming Interface
BPMN	Business Process Model and Notation
CRUD	Create, Read, Update, Delete
CSEP	Canary Sandbox Environment Provisioner, component of PIACERE WP5
CSP	Cloud Service Provider
DevOps	Development and Operations
DevSecOps	Development, Security and Operations
DMC	DOML Model Checker
DMN	Decision Model and Notation
DoA	Description of Action
DOML	DevSecOps Modelling Language, as provided by PIACERE WP3
DMC	DOML Model Checker
EC	European Commission
GA	Grant Agreement to the project
IaC	Infrastructure as Code
ICG	Infrastructural Code Generator, component of PIACERE WP3
IDE	Integrated Development Environment, component of PIACERE WP3
IEC	Infrastructural Elements Catalogue, component of PIACERE WP5
IEM	IaC Execution Manager, component of PIACERE WP5
IOP	IaC Optimization Platform, component of PIACERE WP5
ISR	IaC Scan Runner
KR	Key Result
KR13	Key Result 13 – PIACERE DevSecOps framework – the integration key result that is presented in this deliverable
MC	Monitoring Controller
PMC	Performance Monitoring Controller
PRC	PIACERE Runtime Controller
SH	Self-Healing
SW	Software
TSR	Technical Specification Report (e.g., this document)

Executive Summary

This document is the third version of same-named D2.3 and D2.4. It comes with the same structure as v2 but updated contents and dedicated subsections discussing important differences since the previous version (v2) so that it is easy to see the changes and the document remains self-contained. Unlike v2, there is no appendix on Camunda – that approach has been abandoned and this deliverable reports on the reasoning and the new architecture for the current runtime. In summary, the PRC's role (the main component implementing the PIACERE DevSecOps Framework) has shifted to only mediate in self-healing and focus on the core required functionality.

This document describes the integration of the PIACERE framework v3 as a whole (Key Result 13 [KR13] v3) and acts as the Technical Specification Report (TSR) for the integrated PIACERE framework and the reported new software – PIACERE Runtime Controller (PRC), also in v3. The PRC orchestrates the runtime of the PIACERE framework. The integration options of the IDE are also discussed as the IDE is the central point of PIACERE framework's design time.

The work reported here has been done as part of Task 2.3 (T2.3) in Work Package 2 (WP2) but it involves all other technical WPs, that is WP3, WP4, WP5, and WP6, as they provide the tools that compose the PIACERE framework. The PRC interacts with tools of the PIACERE runtime, i.e., those from WP5 and WP6.

The document starts with an introduction to the contents. Next, it has an explanation of the scope of integration – this lays ground for the rest of the document. It is important to note that this document does not dive into the WP-internal details of integration, e.g., those found mostly in WP6 work but also in parts of other packages. Then, it moves onto the description of integration points considered in this integration work which describe how the integration was conducted. After that, sections on PRC implementation and usage are included which put the PRC in context and describe it in detail. Finally, a conclusions section closes the document.

The main result of this deliverable is the integrated PIACERE framework and the PRC. Together they allow the PIACERE user to reap the benefits of all the PIACERE tooling provided in the first two and half years (i.e., by month 30) of the project and showcase that the PIACERE-provided tools can be used together.

1 Introduction

1.1 About this deliverable

This document contains the details of the integration of the PIACERE framework v3 as a whole (KR13 v) and is the TSR for it and the PRC v3. It also describes the CI/CD effort in the PIACERE project. This is v3 of same-named D2.3 and D2.4.

The integration involves all tools produced as part of other technical WPs, i.e., from WP3 to WP6. The PRC orchestrates the runtime and interacts with runtime tools from WP5 and WP6. The goal of this deliverable is to put the technical results in context and present their details as well as their usage. This v3 summarises also the changes made between v2 and v3 of the tools' interfaces and their integration.

This document does not concern itself with internal integration of components inside their respective deliverables as is the case with WP6 tooling as well as parts of other WPs.

1.2 Document structure

The document is comprised of 7 sections split into subsections. The 1st section is this introduction. The 2nd section describes the scope of integration. The 3rd section goes into detail on the integration points. The 4th section introduces PRC's architecture and technical aspects. The 5th section describes the usage side of PRC. The 6th section offers conclusion. The final 7th section contains references.

2 Scope of integration

This section lays the ground for understanding the integration of the PIACERE DevSecOps framework, also called PIACERE framework or simply PIACERE. The scope is described here along with main points of integration.

2.1 Changes since v2

There are no changes in the scope of integration and thus this section remains mostly unchanged except for references to current versions of components.

2.2 Components used in the integration of v3

In the first version of the PIACERE framework, the components used for integration are all those available at month 30 of the project. They are summarised, along with the respective Key Results and Deliverables in the table below (Table 1 PIACERE Framework v3 components, below). Various sections refer back to these deliverables if the reader wants to know more about the behaviour of a particular tool.

Table 1 PIACERE Framework v3 components

Key Results[1]	Deliverable	Component name
KR13	D2.4 (this one)	PIACERE Runtime Controller (PRC)
KR3	D3.6	Infrastructural Code Generator (ICG)
KR2	D3.9	Integrated Development Environment (IDE)
KR5	D4.3	DOML Model Checker (DMC)
KR6, KR7	D4.6	IaC Scan Runner (ISR)
KR10	D5.3	IaC Execution Manager (IEM)
KR8	D5.6	Canary Sandbox Environment Provisioner (CSEP)
KR9	D5.9	IaC Optimization Platform (IOP)
KR9	D5.9	Infrastructural Elements Catalogue (IEC)
KR11, KR12	D6.3	Monitoring Controller (MC)
KR11	D6.3	Performance Monitoring Controller (PMC)
KR11	D6.3	Performance Self-Learning
KR11	D6.3	Self-Healing (SH)
KR11, KR12	D6.3	Security Monitoring and Self-Learning

All components used for the integration described in this deliverable (except the IDE) have their interfaces follow the popular REST API practice and are described in the OpenAPI [1] format. Certain details on interfaces and the interaction with them are described in section 3 (more details can be found in each deliverable technical report, see Table 1 above). The IDE is a desktop component and thus does not offer a REST API.

2.3 Main points of integration

This section describes two main points of integration in the PIACERE workflow – the IDE and the PRC (PIACERE Runtime Controller). The latter's details are reported in this deliverable.

[1] After a longer inspection, one may notice KR1 and KR4 missing from the table. However, they are DOML (DevSecOps Modelling Language) and its extensions, respectively, and thus are not providing software components for integration.

2.3.1 IDE: Design time to runtime

One of the pivotal tools of the PIACERE project is the IDE – being a Graphical User Interface (GUI) of the PIACERE framework, it supports the design time of the PIACERE Framework and is a gateway to the runtime. Several other Key Results (KRs) of the project are integrated with the IDE using one of integration mechanisms that IDE and these technologies commonly provide. The PIACERE IDE is customized with suitable plug-ins that integrate with the different tools, to minimize the learning curve and simplify the adoption of the PIACERE DevSecOps IaC approach.

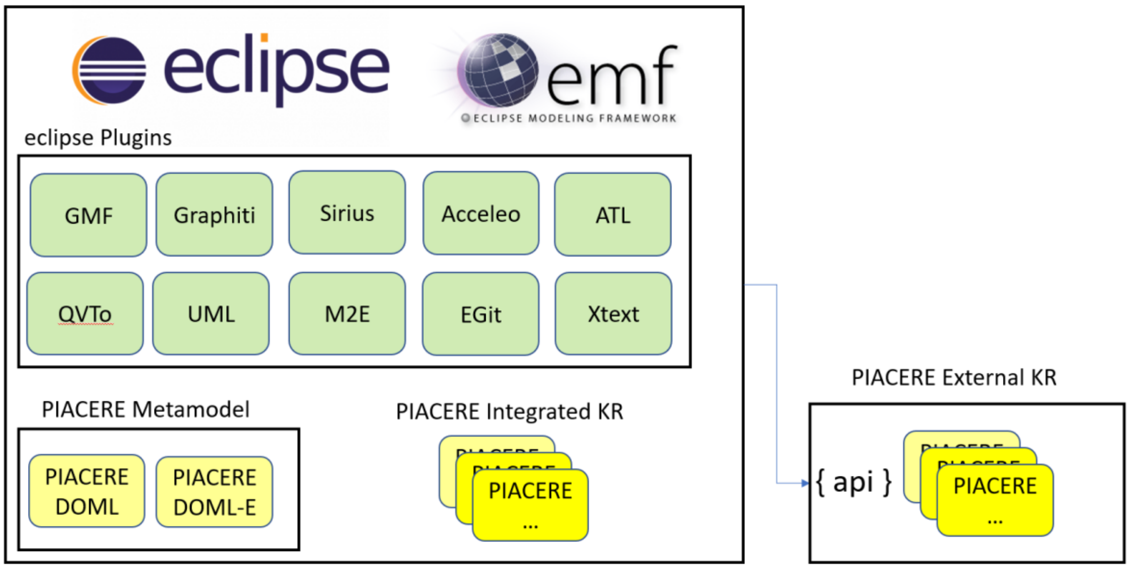


Figure 1 PIACERE IDE integration options

Several integration patterns, focusing on the Eclipse plugin architecture, are available. The proposed ones are:

1. **Native Extension/plugin**. With this option, the tool will be integrated as an Eclipse plugin and will be embedded into the PIACERE IDE:

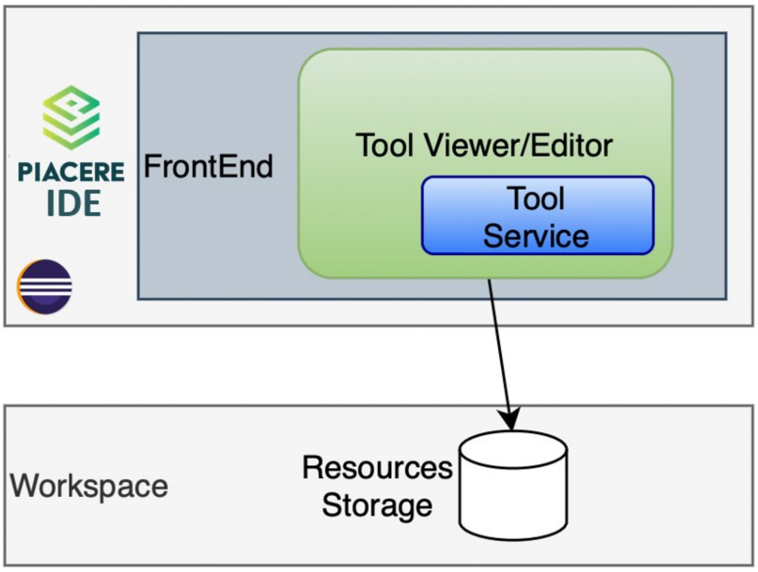


Figure 2 Eclipse Plugin - Integration mechanism

2. **External Service: Backend Service**. This option allows to integrate an external Tool Service which could receive (for example, through a REST API) a DOML model (D3.1) and any necessary configuration to perform some task, e.g., storing its results in a repository.

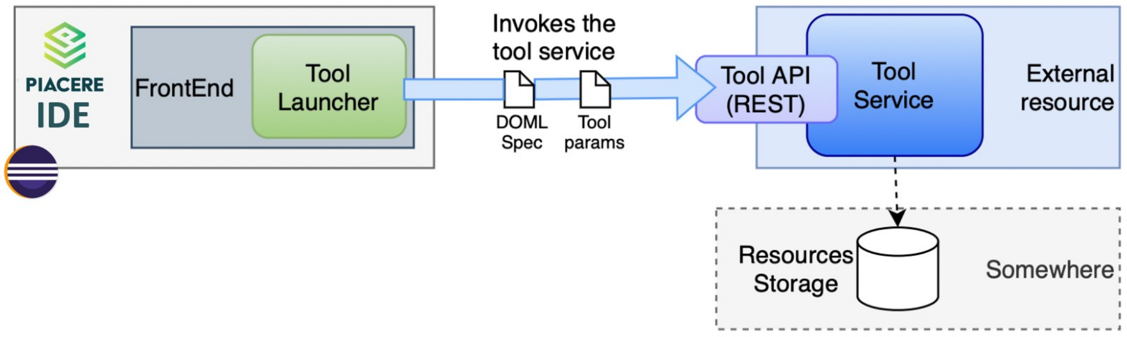


Figure 3 External Service: Backend Service

3. **External Service: Interactive Service**. In this case, the external Tool Service provides a response to the PIACERE IDE (either directly or via a callback) with some tool-generated resources that can then be retrieve and stored by the IDE.

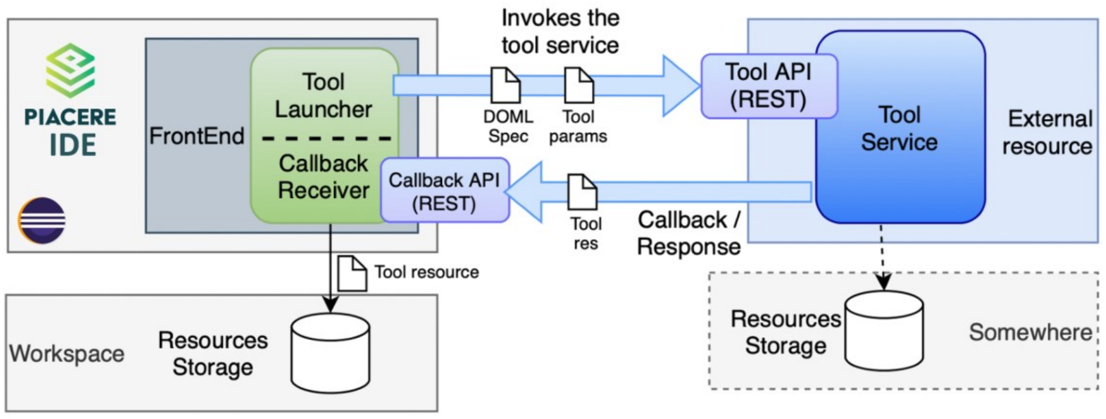


Figure 4 Interactive Service

4. **External Service: Interactive Service with visual (Editor/Viewer) feedback**. This final case is an extension of the previous one where it is required to provide some kind of Eclipse extension to visualize or to manage/edit tool-generated resources.

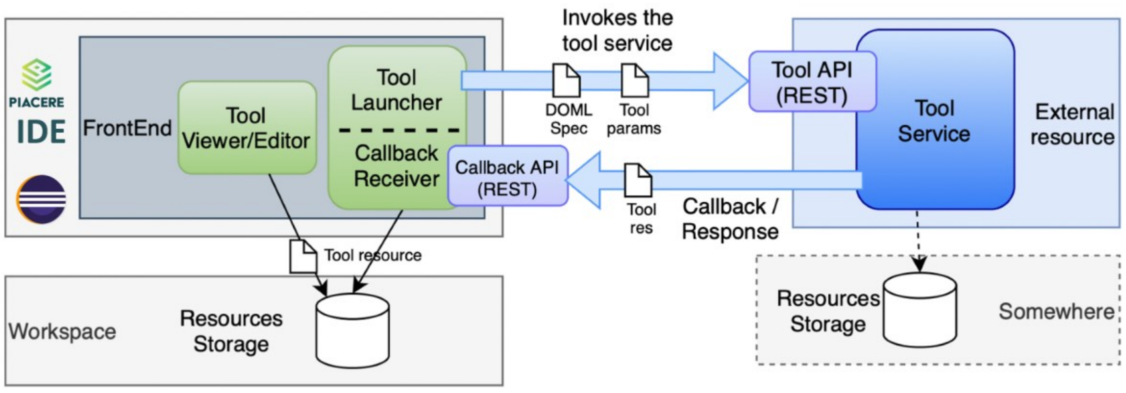
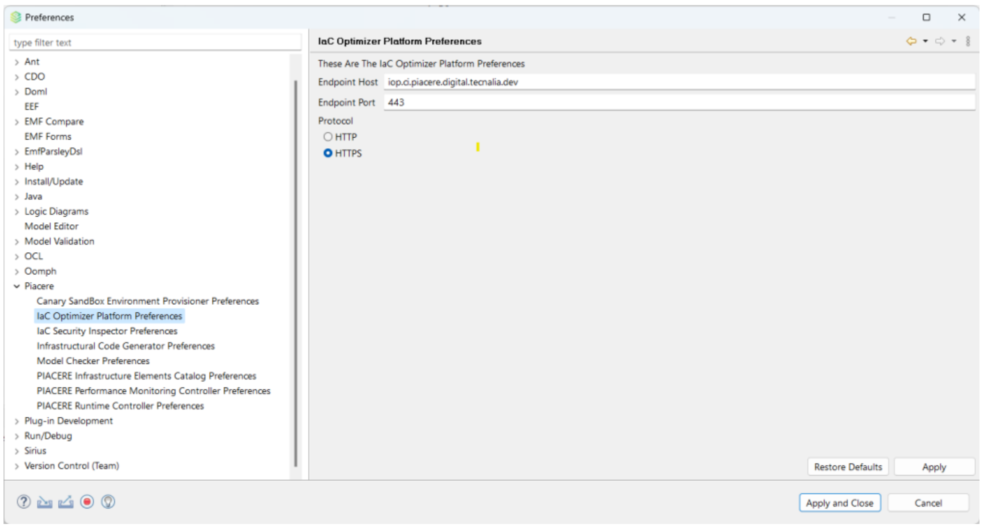


Figure 5 Interactive Service with visual feedback

The PIACERE tools integrated with the IDE can be grouped into two types. Design-time tools, which allow designing/defining the solution to be deployed, and runtime tools, which try to implement the designed system. All PIACERE tools, with the exception of the DOML which has been integrated using pattern 1, have been integrated using a REST approach (as described in the next section "Integrations implementation") which corresponds to patterns 2 and 3 above. Each tool publishes an API well defined using the OpenAPI notation. These APIs are invoked via the different menus and submenus of the IDE and the endpoints are configurable. Some tools require a visual interface to interact with or to show the results in a human-friendly manner (i.e., follow pattern 4 above). These interfaces have been developed and included in the IDE and are described in the next section "Integrations implementation".


Preferences

type filter text

- Ant
- CDO
- Doml
- EEF
- EMF Compare
- EMF Forms
- EmfParsleyDsl
- Help
- Install/Update
- Java
- Logic Diagrams
- Model Editor
- Model Validation
- OCL
- Oomph
- Piacere
  - Canary SandBox Environment Provisioner Preferences
  - IaC Optimizer Platform Preferences
  - IaC Security Inspector Preferences
  - Infrastructural Code Generator Preferences
  - Model Checker Preferences
  - PIACERE Infrastructure Elements Catalog Preferences
  - PIACERE Performance Monitoring Controller Preferences
  - PIACERE Runtime Controller Preferences
- Plug-in Development
- Run/Debug
- Sirius
- Version Control (Team)

IaC Optimizer Platform Preferences

These Are The IaC Optimizer Platform Preferences

Endpoint Host: iop.ci.piacere.digital.tecnalia.dev

Endpoint Port: 443

Protocol: HTTP [✓ HTTPS]

Figure 6 IDE preferences menu for PIACERE

2.3.2 PRC: Control of the runtime

The other pivotal tool of the PIACERE project is the PRC – PIACERE Runtime Controller – which orchestrates the runtime part of the PIACERE framework and is a bridge for the IDE to go from design time to runtime. Various runtime components are interacted with and interact with the PRC. The details of PRC are reported in this deliverable in sections 4 and 5. This section is kept here short only to maintain the structure.

2.4 Delivery of the integrated components

All components were developed concurrently and their code history was tracked using the Git version control system. The **y3** branch tracks the progress for year 3 of the PIACERE project. All components, except the IDE, are containerised and were co-installed on the project-internal integration server. The Dockerfile [2] and docker-compose [3] recipes are available for easy recreation of the integration environment. Each component has a dedicated internal network which co-hosts the various subcomponents the tool might have. All REST API endpoints are exposed on a common network to allow for communication between the components. Access multiplexing is done with Traefik [4]. The PIACERE framework installation is done entirely using docker-compose.

2.5 Continuous Integration / Continuous Delivery (CI/CD)

In addition to the "point releases" and their deployments maintained for demos and manual testing and experimentation, the PIACERE project has adopted a CI/CD strategy that validates each change to any of the PIACERE subcomponents. This is done with the help of GitLab CI/CD. Each Merge Request (MR) runs a set of tests (code linting, container image building, unit tests, integration tests) and their results determine whether the MR can be merged. The integration test deploys the entire centralised PIACERE framework (i.e., all components except the IDE which runs on user's desktop) and runs simple flows against the deployed solution using example DOML documents and the OpenStack cloud deployed by PIACERE Canary Sandbox Environment Provisioner (CSEP). This ensures that the components work in tandem: that they can be installed together, run together and, most importantly, used together in a coherent manner as defined by the PIACERE project.

3 Integrations implementation

This section describes the implementations of integration points – between main integration points and the other tools that are invoked or invoke the main integration points. The order of components in section names is: *invoker – invoked*. The invoked component is described in the context of being invoked by the invoker.

3.1 Changes since v2

Most integrations have had at least minor changes between v2 and v3. With respect to IDE's integrations, the two static verification tools, the DOML Model Checker and the IaC Scan Runner, have had their interfaces enhanced to output a user-friendly web page rendering of the analysis results. On the side of PRC, the PRC's role has shifted to not handle self-healing internally but only mediate in its realisation thanks to the centralised role and access to deployed IaC and necessary credentials.

3.2 IDE&IOP – IEC

The Infrastructural Elements Catalogue (IEC) stores information about the services available at service providers, and classify them based on their properties. as well as the instances of each of these services deployed by the PIACERE infrastructure. The Catalogue is also expected to receive and offer information about the status of the deployed instances.

Behaviour

The IEC is part of the optimization component and is not intended to be used directly by the user. Instead, it is used behind the scenes (by the IOP and the IDE mainly). This use is made via the APIs provided by the IEC and is transparent to the final user.

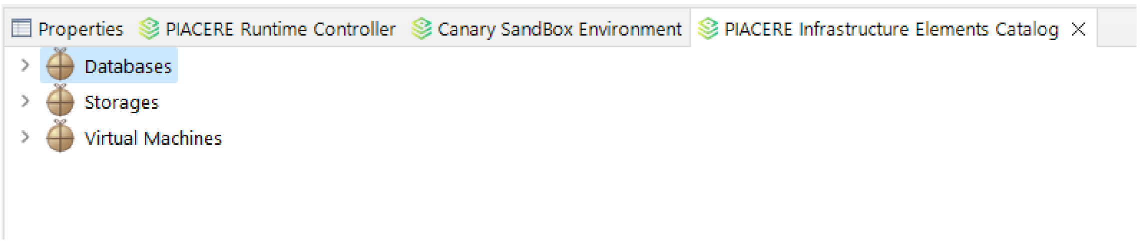
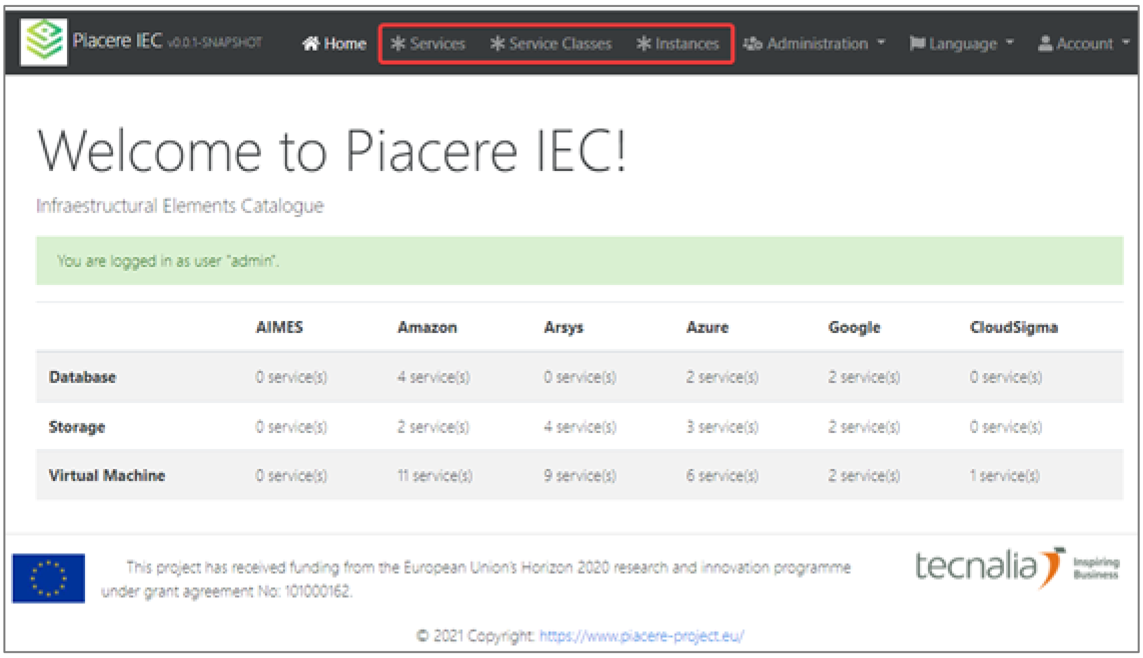

Properties
PIACERE Runtime Controller
Canary SandBox Environment
PIACERE Infrastructure Elements Catalog
Databases
Storages
Virtual Machines

Figure 7 IDE - IEC tab

However, the Catalogue provides a GUI (Graphical User Interface) that allows user interaction to check and edit the contained information. The GUI enables the create, read, update and delete (CRUD) capabilities for several entities: services, service classes and instances (Figure 8).



Welcome to Piacere IEC!

Infraestructural Elements Catalogue

You are logged in as user "admin".

	AIMES	Amazon	Arsys	Azure	Google	CloudSigma
Database	0 service(s)	4 service(s)	0 service(s)	2 service(s)	2 service(s)	0 service(s)
Storage	0 service(s)	2 service(s)	4 service(s)	3 service(s)	2 service(s)	0 service(s)
Virtual Machine	0 service(s)	11 service(s)	9 service(s)	6 service(s)	2 service(s)	1 service(s)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No: 101000162.

© 2021 Copyright: https://www.piacere-project.eu/

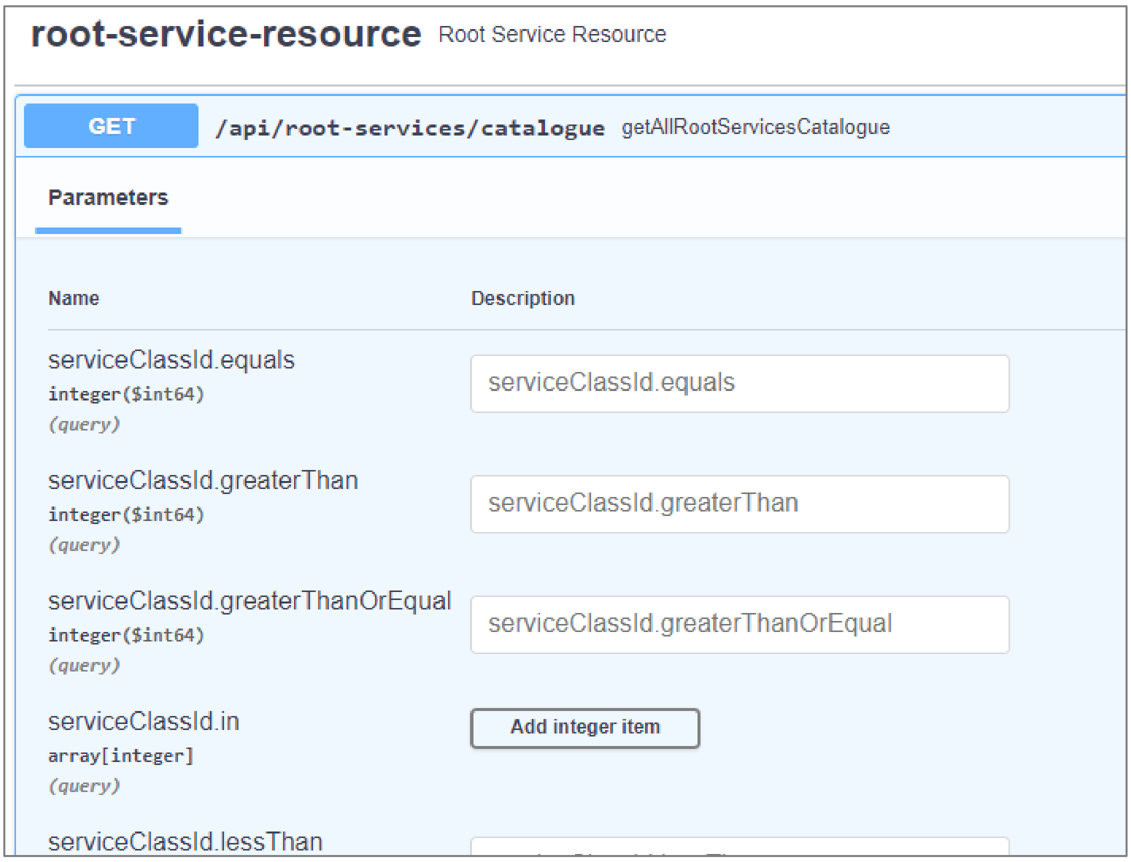
Figure 8 IEC web Graphical User Interface

The IEC receives information from different sources. What we have called "static information", concerns the available services provided by the CSPs. This information is pre-loaded in the IEC database using an SQL script.

The IEC also provides a REST API that can be accessed for automation purposes.

Input

The interaction between the IDE and the IEC is performed through the API. The REST API enables the interaction through a single endpoint (*/catalogue*). Currently, we include only one method that is called by the IDE – GET /api/root-serv– GET /api/root-service/catalogue (see Figure 9) – to obtain all the services present in the IEC.


root-service-resource Root Service Resource
GET
/api/root-services/catalogue getAllRootServicesCatalogue
Parameters
Add integer item

Name	Description
serviceClassId.equals integer($int64) (query)	serviceClassId.equals
serviceClassId.greaterThan integer($int64) (query)	serviceClassId.greaterThan
serviceClassId.greaterThanOrEqual integer($int64) (query)	serviceClassId.greaterThanOrEqual
serviceClassId.in array[integer] (query)	Add integer item
serviceClassId.lessThan	

Figure 9 IEC REST API

Output

The output that receives the IDE will be the list of services in a json format. In the next figure, there can be seen a small excerpt of the response, showing part of the “C1_Spain” service of type “Virtual Machine”.

```
[
  {
    "id": 1,
    "serviceName": "C1_Spain",
    "deletedDate": null,
    "serviceClass": {
      "id": 3,
      "serviceClassName": "Virtual Machine",
      "deletedDate": null
    },
    "serviceAttributeValues": [
      {
        "id": 1,
        "serviceAttributeValue": "00EU",
        "unitValue": "",
        "serviceAttributeType": {
          "id": 1,
          "name": "Region",
          "nfrName": "Region",
          "isEnumeration": true,
          "isForm": false,
          "isCommon": true,
          "isFunctionalRequirement": false,
          "units": "",
          "unitFactor": "",
          "unitRule": "LIKE",
          "evalRule": ""
        }
      },
      {
        "id": 2,
        "serviceAttributeValue": "SPEU",
        "unitValue": "",
        "serviceAttributeType": {
          "id": 2,
          "name": "Zone",
          "nfrName": "Zone",
          "isEnumeration": true,
          "isForm": false,
          "isCommon": true,
          "isFunctionalRequirement": false,
          "units": "",
          "unitFactor": "",
          "unitRule": "LIKE",
          "evalRule": ""
        }
      },
(…)
```

Figure 10 Excerpt of the output response to the IDE call

The integrated PIACERE IDE gathers this information and offers a way of using the content of the catalogue through a specific view (the “IEC View”), where the services are listed, grouped by the element type (a further click on any of the services shows its properties).

3.3 IDE – DOML Model Checker

The DOML Model Checker (DMC) is a tool that statically analyses IaC for compliance with a pre-defined or user-supplied specification. It reports to the user whether the specification is satisfied or not. It now also includes a tool to verify Cloud Service Providers compatibility by checking a selection of properties specified in DOML against a set of valid values.

Behaviour

The user requests the verification of the currently displayed DOML model in the IDE through an appropriate user interaction (e.g., a button or menu entry). Then, the IDE sends to the DMC a model checking request containing the DOML model. The DMC then processes the IaC and sends the outcome of model checking back to the IDE. Finally, the IDE prompts the user to save the verification outcome as a HTML file that can be viewed to inspect the results.

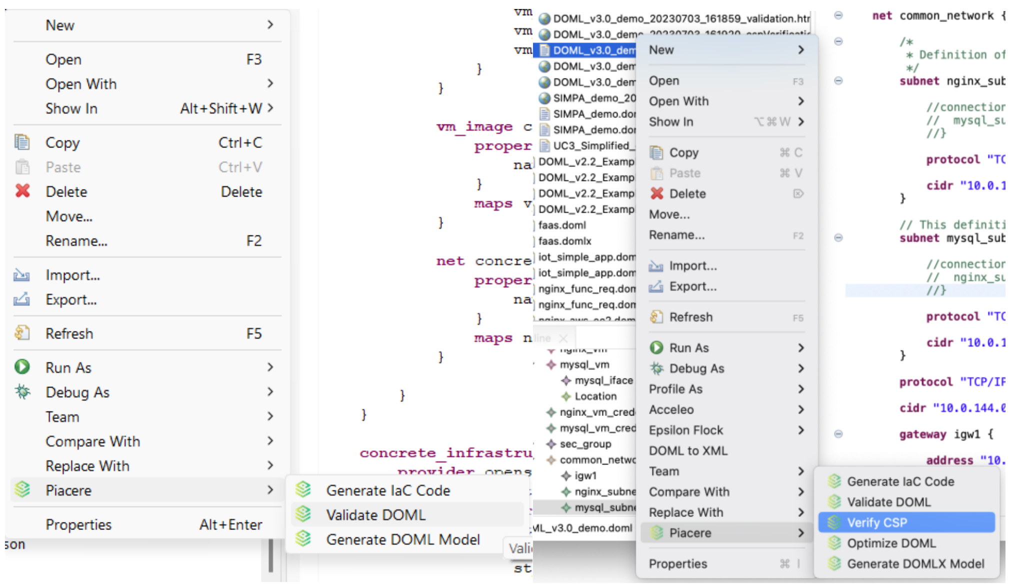


Figure 11 DOML model checker menu option

The interaction between the IDE and the DMC is performed through REST APIs. The REST API enables the interaction by providing a set of endpoints, which accept a POST request containing the DOML model in the DOMLX format. The endpoints /modelcheck and /csp provide a response in JSON format, while the /modelcheck_html and /csp_html provide one in HTML, to allow for a human-readable document when opened in a web browser. The result (described in the Output section) of the model checking process is then sent to the client as the answer to its POST request (Figure 12 DMC REST API). The HTML version of the results includes tables and cards detailing missing or invalid properties of the DOML.

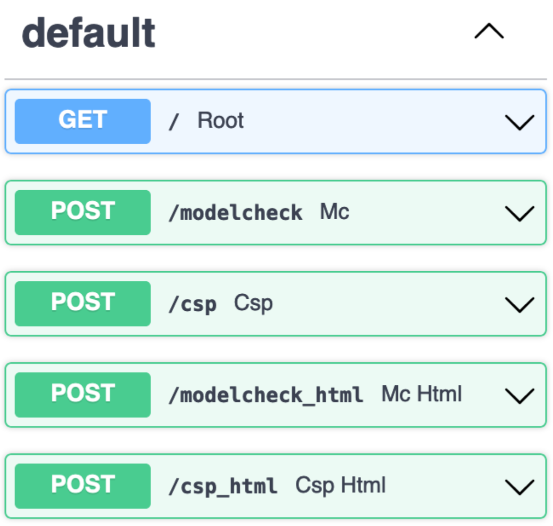


Figure 12 DMC REST API

Input

The DMC receives from the IDE a verification request containing:

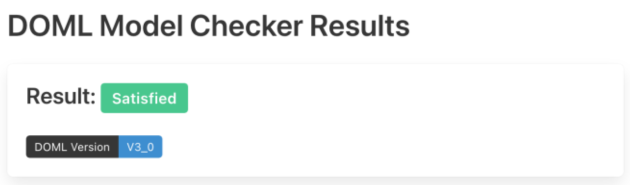
- The DOML model to be verified in in DOMLX/XMI format (required).

Output

In JSON endpoints, the DMC answers with:

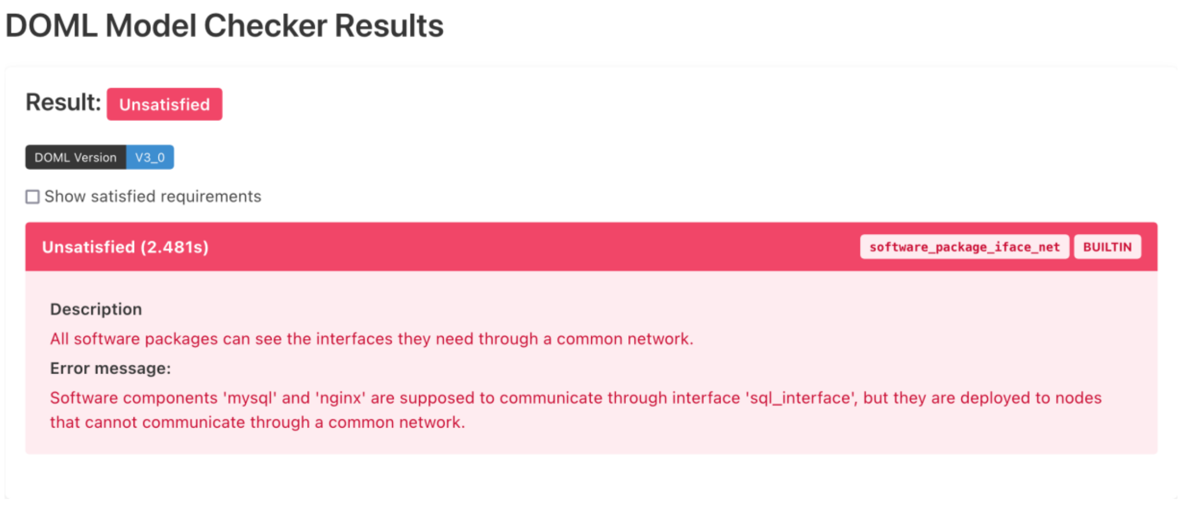
- A string stating the verification outcome (**required**). This can be either "sat", meaning the model satisfies the requirements, "unsat", meaning the model does not satisfy the requirements, or "dontknow", meaning the model checking process was inconclusive due to a timeout.
- A string describing the issue(s) found in the model (**optional**, set if there are issues).

Alternatively, the output of HTML endpoints is shown below (Figure 13 Model checker and CSP Compatibility Tool output):


DOML Model Checker Results

Result: Satisfied

DOML Version V3_0


DOML Model Checker Results

Result: Unsatisfied

DOML Version V3_0

Show satisfied requirements

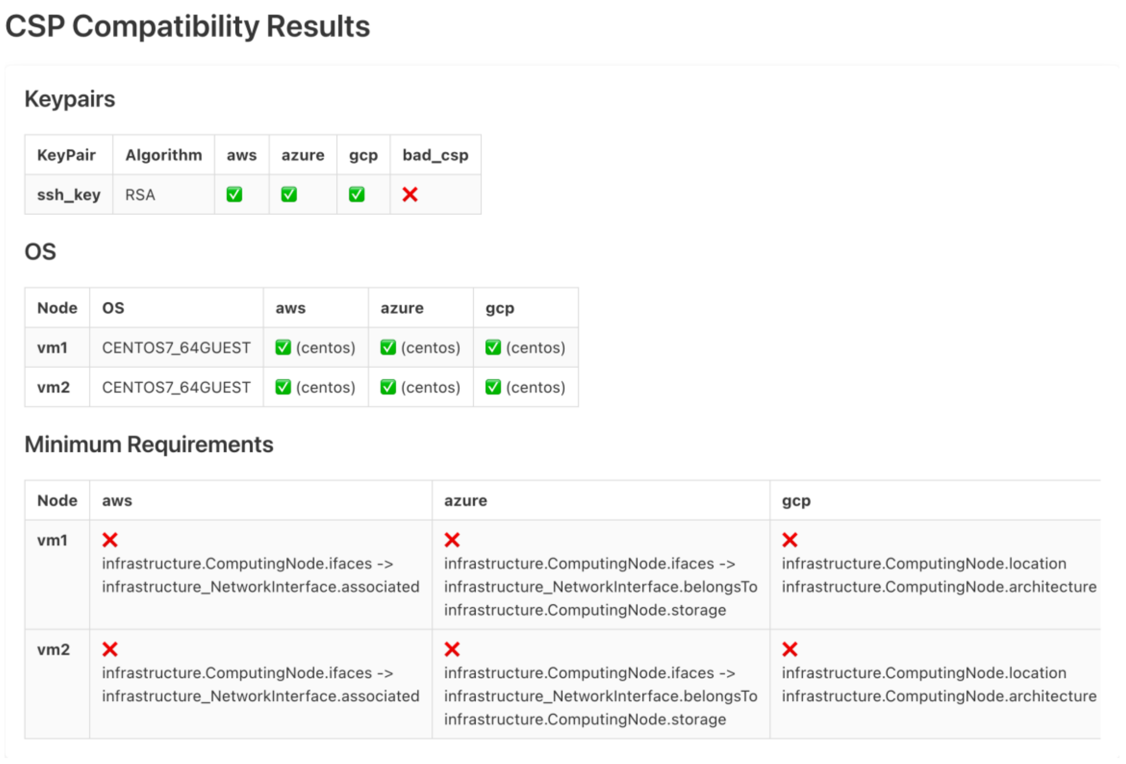
Unsatisfied (2.481s) software_package_iface_net BUILTIN

Description

All software packages can see the interfaces they need through a common network.

Error message:

Software components 'mysql' and 'nginx' are supposed to communicate through interface 'sql_interface', but they are deployed to nodes that cannot communicate through a common network.



CSP Compatibility Results

Keypairs

KeyPair	Algorithm	aws	azure	gcp	bad_csp
ssh_key	RSA	✅	✅	✅	❌

OS

Node	OS	aws	azure	gcp
vm1	CENTOS7_64GUEST	✅ (centos)	✅ (centos)	✅ (centos)
vm2	CENTOS7_64GUEST	✅ (centos)	✅ (centos)	✅ (centos)

Minimum Requirements

Node	aws	azure	gcp
vm1	❌ infrastructure.ComputingNode.ifaces -> infrastructure_NetworkInterface.associated	❌ infrastructure.ComputingNode.ifaces -> infrastructure_NetworkInterface.belongsTo infrastructure.ComputingNode.storage	❌ infrastructure.ComputingNode.location infrastructure.ComputingNode.architecture
vm2	❌ infrastructure.ComputingNode.ifaces -> infrastructure_NetworkInterface.associated	❌ infrastructure.ComputingNode.ifaces -> infrastructure_NetworkInterface.belongsTo infrastructure.ComputingNode.storage	❌ infrastructure.ComputingNode.location infrastructure.ComputingNode.architecture

Figure 13 Model checker and CSP Compatibility Tool output

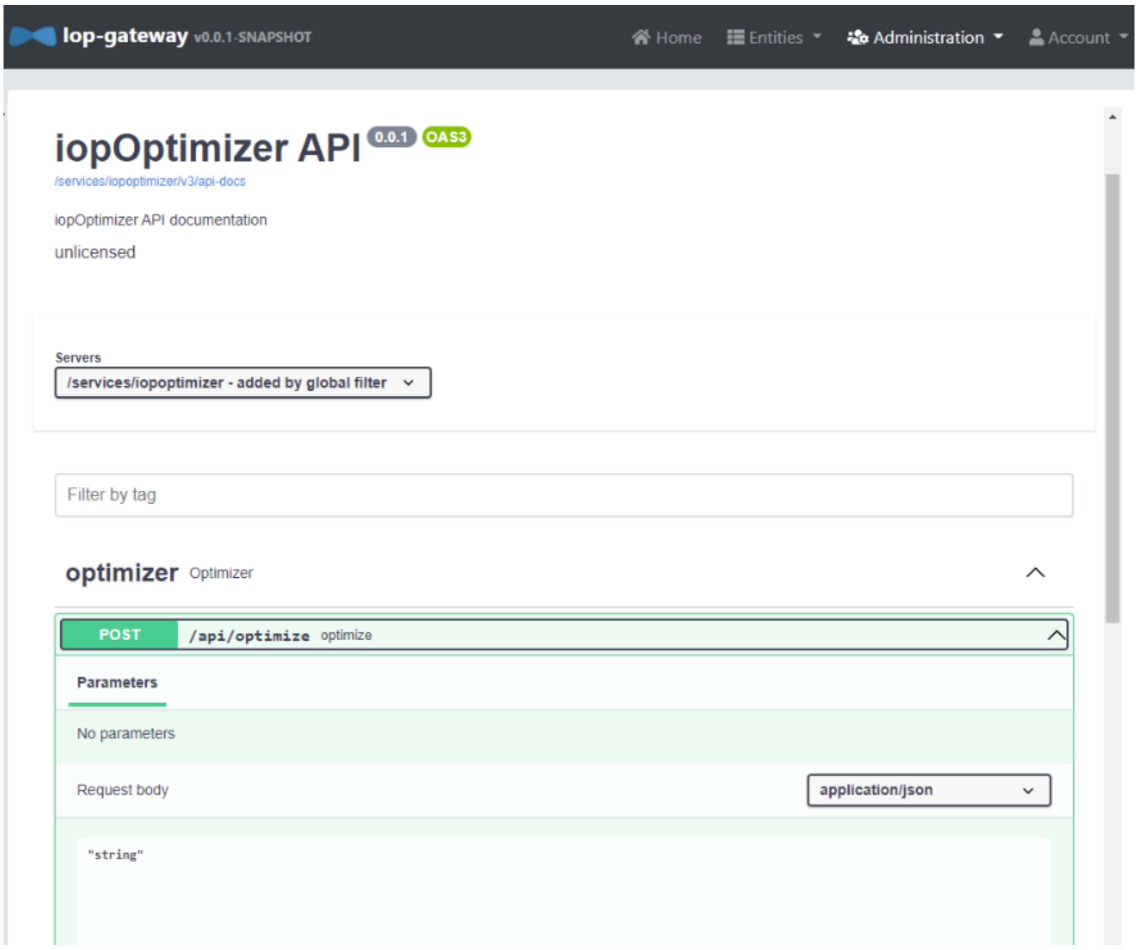
3.4 IDE – IOP

The IaC Optimization Platform is a tool for optimizing the IaC based on the constraints in the DOML model.

Behaviour

In a nutshell, the IOP is responsible for finding the best possible infrastructure given the input data received. In other words, the optimizer uses an optimization algorithm to find an optimized deployment configuration of the IaC on the appropriate infrastructural elements that best meet the predefined constraints, i.e., are optimal. This information is supposed to be presented to the PIACERE user.

The IOP offers an embedded Swagger UI (Figure 14) to facilitate the understanding and usage by the IDE. Furthermore, the IOP can also be called from the IDE as depicted in Figure 15.



iopOptimizer API 0.0.1 OAS3
/services/iopoptimizer/v3/api-docs

iopOptimizer API documentation

unlicensed

Servers: /services/iopoptimizer - added by global filter

optimizer Optimizer

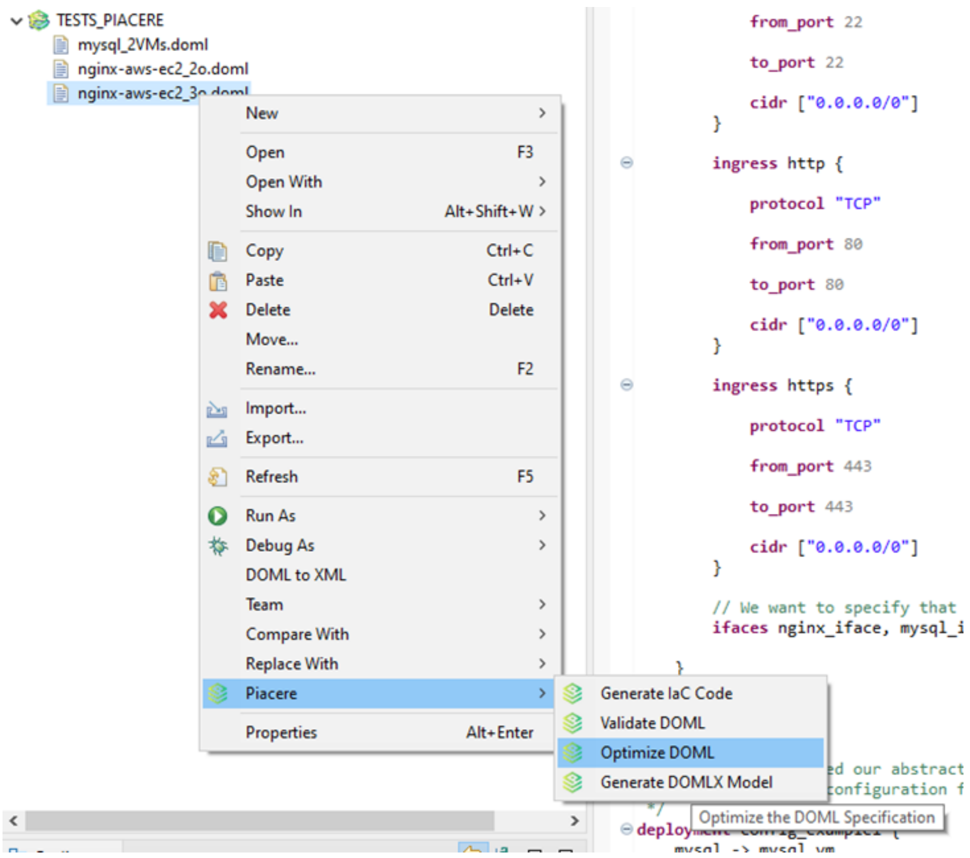
POST /api/optimize optimize

No parameters

Request body: application/json

```
"string"
```

Figure 14 Swagger UI page for IOP



```
        from_port 22
        to_port 22
        cidr ["0.0.0.0/0"]
    }
    ingress http {
        protocol "TCP"
        from_port 80
        to_port 80
        cidr ["0.0.0.0/0"]
    }
    ingress https {
        protocol "TCP"
        from_port 443
        to_port 443
        cidr ["0.0.0.0/0"]
    }
    // We want to specify that
    ifaces nginx_iface, mysql_i
  }
ed our abstract
configuration f
*/
deploy
mysql -> mysql_vm
```

Figure 15 How the IOP is called from the IDE

Input

In v3, the input data is accepted in the DOML 3.0 format.

Output

Once the IOP completes the optimization, it returns as output an updated DOML model.

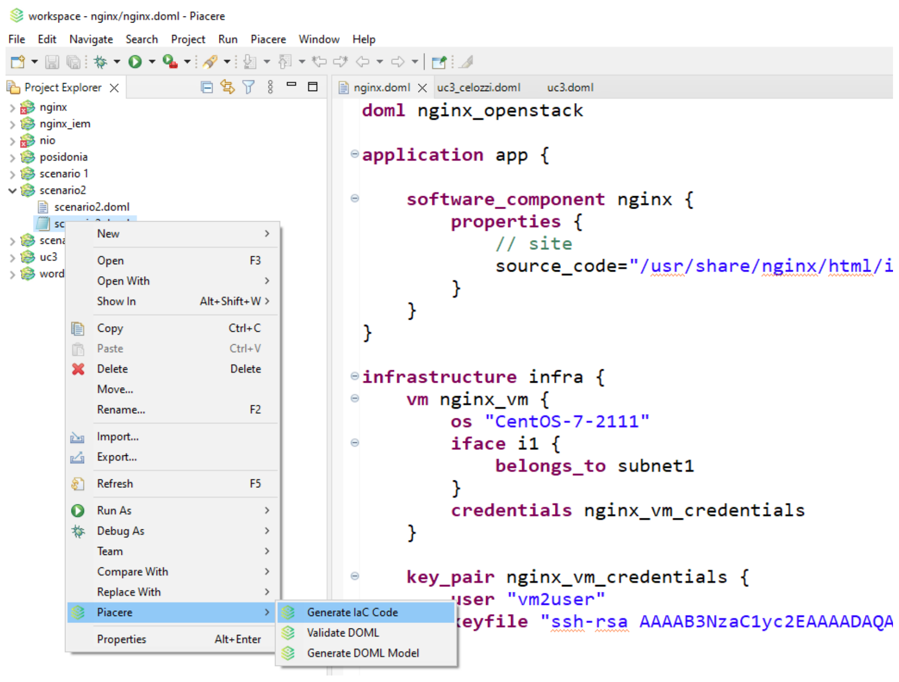
For details on IOP's behaviour, please see D5.9.

3.5 IDE – ICG

The Infrastructural Code Generator (ICG) is the PIACERE component that allows generating executable infrastructural code (IaC) from models written in DOML.

Behaviour

The ICG communicates with the IDE through REST API. The user can request the conversion of the DOML model in IaC code by the IDE menu selection "Generate IaC Code", as shown in Figure 15. After selecting the code generation, the user will be asked for two prompts before receiving the output IaC, one being the request for the asset folder that can contain custom code that the user needs and the second is the name to give to the output package.



```
doml nginx_openstack

application app {

    software_component nginx {
        properties {
            // site
            source_code="/usr/share/nginx/html/i
        }
    }
}

infrastructure infra {
    vm nginx_vm {
        os "CentOS-7-2111"
        iface i1 {
            belongs_to subnet1
        }
        credentials nginx_vm_credentials
    }

    key_pair nginx_vm_credentials {
        user "vm2user"
        eyfile "ssh-rsa AAAAB3NzaC1yc2EAAAADAQA
```

Figure 16 IDE - ICG "Generate IaC Code" menu option

If the conversion is selected, the ICG expects a POST (Figure 17) call from the IDE containing in the body of the call a compressed (.zip extension) package, inside the package there should be: the metamodel with the information; the ecore used to generate the metamodel; the asset folder containing all the user custom code to be added. After the ICG has generated all the IaC files they will be sent back as a response (also compressed in a .zip package) and will be available to the user through the IDE.

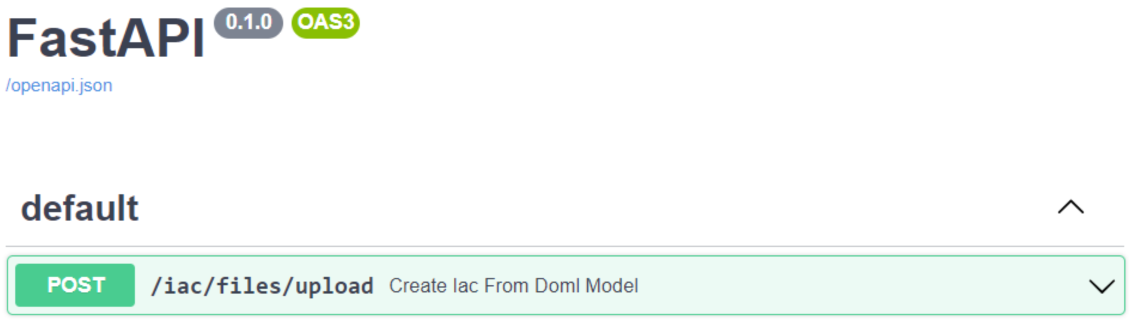


Figure 17 ICG REST API

The ICG also provides logs related to the process of generation of the IaC files, containing errors, warnings and info useful to the end user. These logs need to be exported to the user in a way that is easily readable and comprehensible. Complementary to the generated IaC code, it is possible to generate the Gaia-X[2] compliant self-description file, which includes all data necessary for publishing the application in a Gaia-X Federated Catalogue. This means that results of PIACERE design-time services and applications are digestible not only by the PIACERE runtime framework (PRC, IEM, etc.), but also available to the wider European Gaia-X community, to deploy them with the XFSC orchestrator or other connected services. The PIACERE provides the

[2] https://gaia-x.eu/

initial template for describing the IaC Application and allow Gaia-X users to find and deploy applications on to their infrastructure.

Lastly, the ICG uses configuration files on the local filesystem that allow the extensibility of the tool through the definition of the components and templates, which are going to be expanded when new components are added. To update these files, a new docker image with the update files can be build and the service restarted.

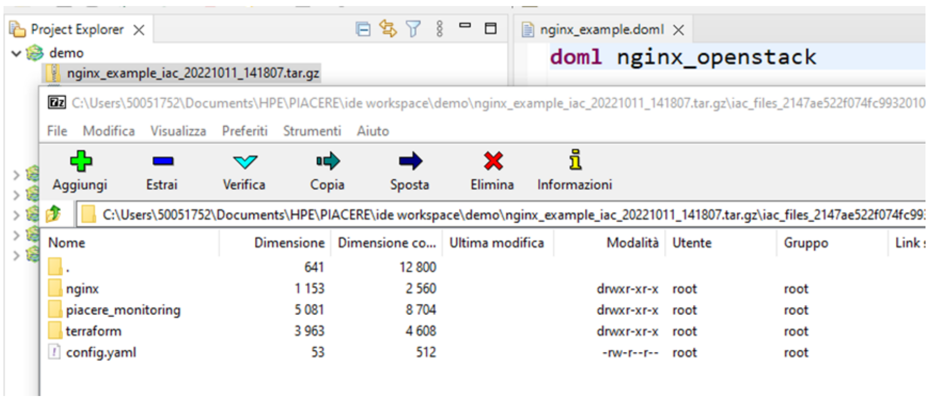
Further details about the ICG component can be found in deliverable D3.6.

Inputs

The input of the ICG is a compressed package (.zip extension) containing the DOMLx model, the ecore and the asset folder.

Outputs

The output of the ICG is a compressed package (.zip extension) containing the IaC files. Beside the IaC files, the IDE offers us the generation of Self-description file – a meta file that provides necessary information to publish the IaC resource in the Gaia-X catalogue.



```
doml nginx_openstack
```

C:\Users\50051752\Documents\HPE\PIACERE\ide workspace\demo\nginx_example_iac_20221011_141807.tar.gz\iac_files_2147ae522f074fc9932010

C:\Users\50051752\Documents\HPE\PIACERE\ide workspace\demo\nginx_example_iac_20221011_141807.tar.gz\iac_files_2147ae522f074fc99

Nome	Dimensione	Dimensione co...	Ultima modifica	Modalità	Utente	Gruppo	Link :
.	641	12 800					
nginx	1 153	2 560		drwxr-xr-x	root	root	
piacere_monitoring	5 081	8 704		drwxr-xr-x	root	root	
terraform	3 963	4 608		drwxr-xr-x	root	root	
config.yaml	53	512		-rw-r--r--	root	root	

Figure 18 ICG IaC generated folder

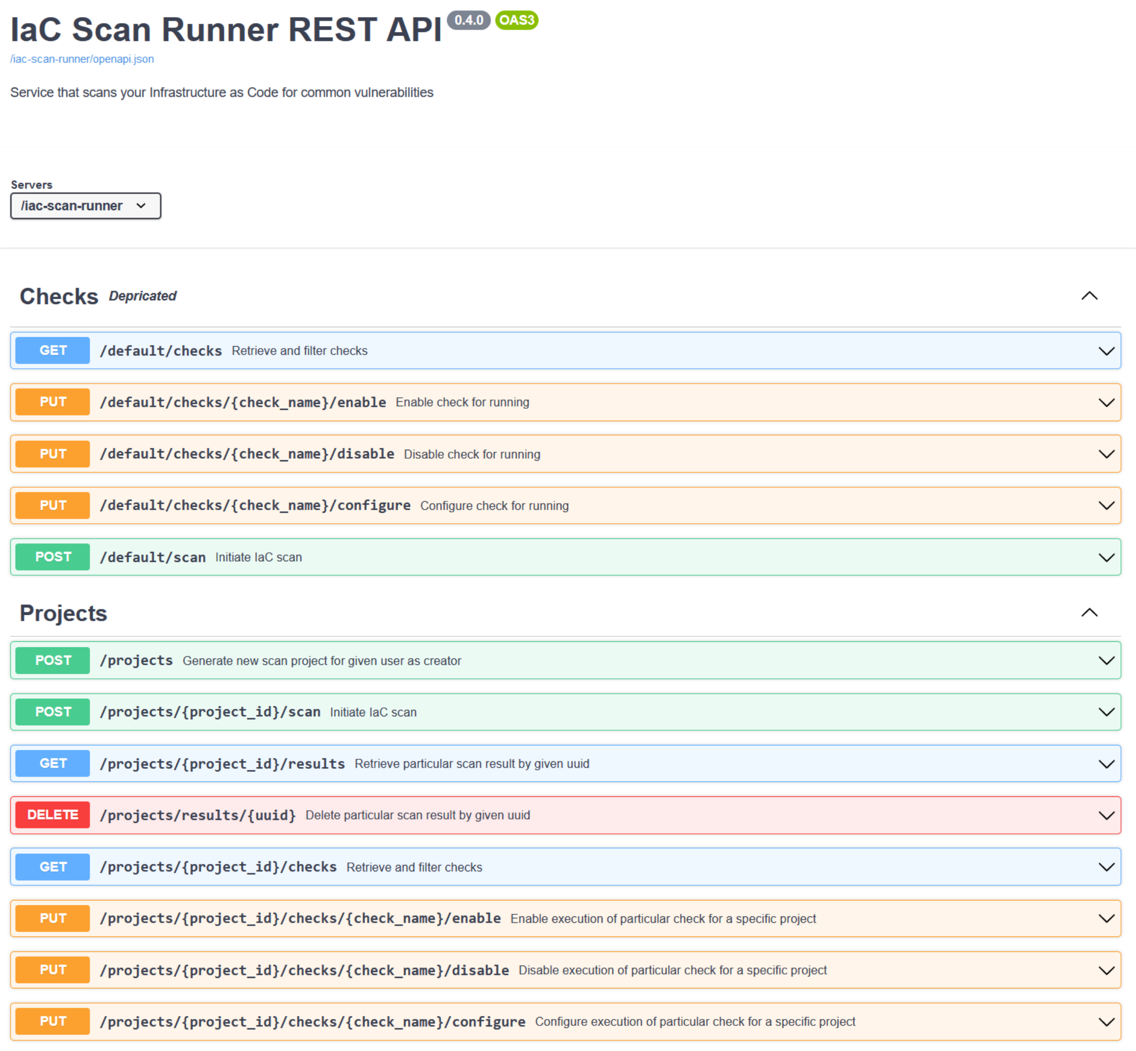
3.6 IDE – IaC Scan Runner

The IaC Scan Runner is a component providing a single interface to the functionalities of IaC Security Inspector and Component Security Inspector. This component includes multiple IaC and component checks that are used to perform scanning of IaC and returning the discovered common security vulnerabilities and misconfigurations.

Behaviour

IaC Scan Runner is usually set up in a Docker container and gets exposed as a REST API service that comes with OpenAPI Specification that allows users to interact with the API through browser (Figure 19). There is a big difference from previous versions - API is now divided into two sections. First section includes test endpoints which are used to test all available checks – this approach was used in previous editions of IDE. Using the test*/checks* endpoint, the API enables the user to list and filter the supported checks (those are described thoroughly within IaC Scanner documentation: https://xlab-si.github.io/iac-scanner-docs/). Apart from IaC and component distinctions, checks can also be local (executing on the machine where IaC Scan Runner is running) or remote (executing partially or wholly on the remote service, e.g., Snyk, SonarCloud, etc.). Checks can be enabled (using *test/enable* endpoint) or disabled (using *test/disable* endpoint) for running. Some of them are disabled by default and need to be configured (via *test/configure* endpoint by supplying necessary configuration files or/and credentials) before running. When configuration steps are done, checks can be run by calling *test/scan*, which expects the compressed IaC file and a list of checks to be executed while scanning (if no list is supplied, then API runs all enabled checks).

Second section, named Project, serves as per project configured checks. That means that user can define specific configuration per project, and this is a proposed way to allow co-existence of multiple scans and temporary storing all results for some time. Project is created by calling *POST /project.* Checks can be enabled, disabled, and configured in the same way as it is described in first section of the API. It is also possible, to set project configuration beforehand with appropriate JSON file (an example JSON can be seen in [https://xlab-si.github.io/iac-scanner-docs/)](https://xlab-si.github.io/iac-scanner-docs/). Once appropriate configuration file is created you can create new configuration instance by calling */project/configuration* and set its content by uploading your configuration file to *project/configuration/parameters.* Configuration then needs to be bind to a project, calling */project/configuration/bind* endpoint. Once everything is configured you can initiate scan by calling */project/scan.* Each scan result is now saved and can be viewed by */project/results* or deleted by */project/results/{result_uuid}.*


IaC Scan Runner REST API 0.4.0 OAS3
/iac-scan-runner/openapi.json

Service that scans your Infrastructure as Code for common vulnerabilities

Servers
/iac-scan-runner

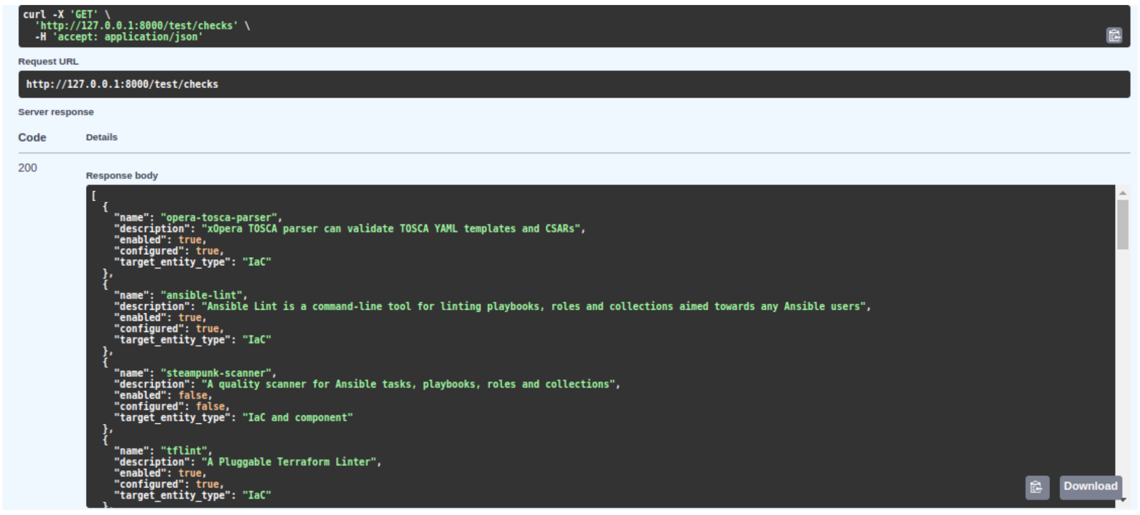
Checks *Depricated*

GET `/default/checks` Retrieve and filter checks
PUT `/default/checks/{check_name}/enable` Enable check for running
PUT `/default/checks/{check_name}/disable` Disable check for running
PUT `/default/checks/{check_name}/configure` Configure check for running
POST `/default/scan` Initiate IaC scan

Projects

POST `/projects` Generate new scan project for given user as creator
POST `/projects/{project_id}/scan` Initiate IaC scan
GET `/projects/{project_id}/results` Retrieve particular scan result by given uuid
DELETE `/projects/results/{uuid}` Delete particular scan result by given uuid
GET `/projects/{project_id}/checks` Retrieve and filter checks
PUT `/projects/{project_id}/checks/{check_name}/enable` Enable execution of particular check for a specific project
PUT `/projects/{project_id}/checks/{check_name}/disable` Disable execution of particular check for a specific project
PUT `/projects/{project_id}/checks/{check_name}/configure` Configure execution of particular check for a specific project

Figure 19 Section from Swagger UI page for IaC Scan Runner REST API



```
curl -X 'GET' \
  'http://127.0.0.1:8000/test/checks' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/test/checks
```

Server response

Code Details

200 Response body

```
[
  {
    "name": "opera-tosca-parser",
    "description": "xOpera TOSCA parser can validate TOSCA YAML templates and CSARs",
    "enabled": true,
    "configured": true,
    "target_entity_type": "IaC"
  },
  {
    "name": "ansible-lint",
    "description": "Ansible Lint is a command-line tool for linting playbooks, roles and collections aimed towards any Ansible users",
    "enabled": true,
    "configured": true,
    "target_entity_type": "IaC"
  },
  {
    "name": "steampunk-scanner",
    "description": "A quality scanner for Ansible tasks, playbooks, roles and collections",
    "enabled": false,
    "configured": false,
    "target_entity_type": "IaC and component"
  },
  {
    "name": "tflint",
    "description": "A Pluggable Terraform Linter",
    "enabled": true,
    "configured": true,
    "target_entity_type": "IaC"
```

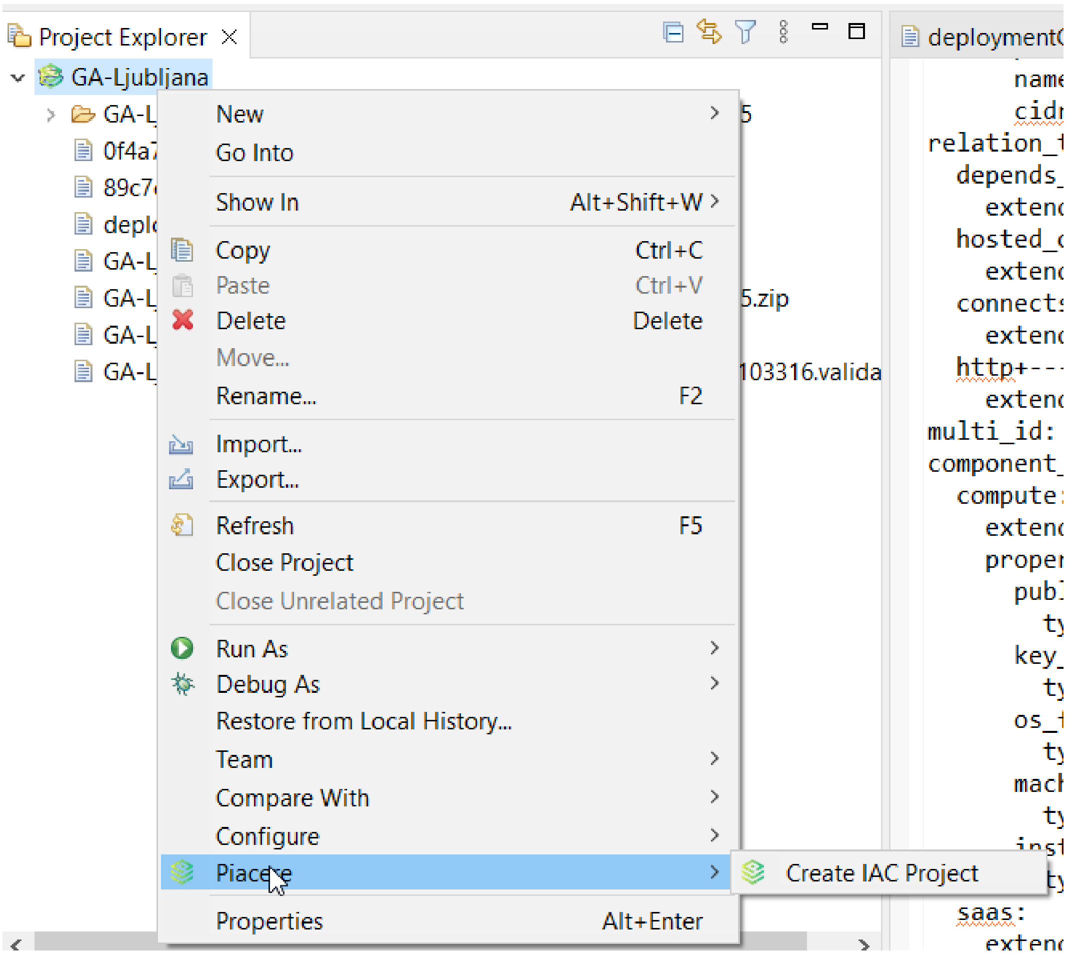
Figure 20 Filtering out check for Terraform in the IaC Scan Runner REST API

IDE Integration of IaC Scan Runner

The IaC scan is performed from the IDE just after the ICG generates IaC code from the DOML. The idea is that a user is able to locate the IaC code within the IDE and, from the context menu, send it directly to the IaC Scan Runner Service. The output of the IaC scan runner will be a text that is retrieved by the IDE and presented to the user.

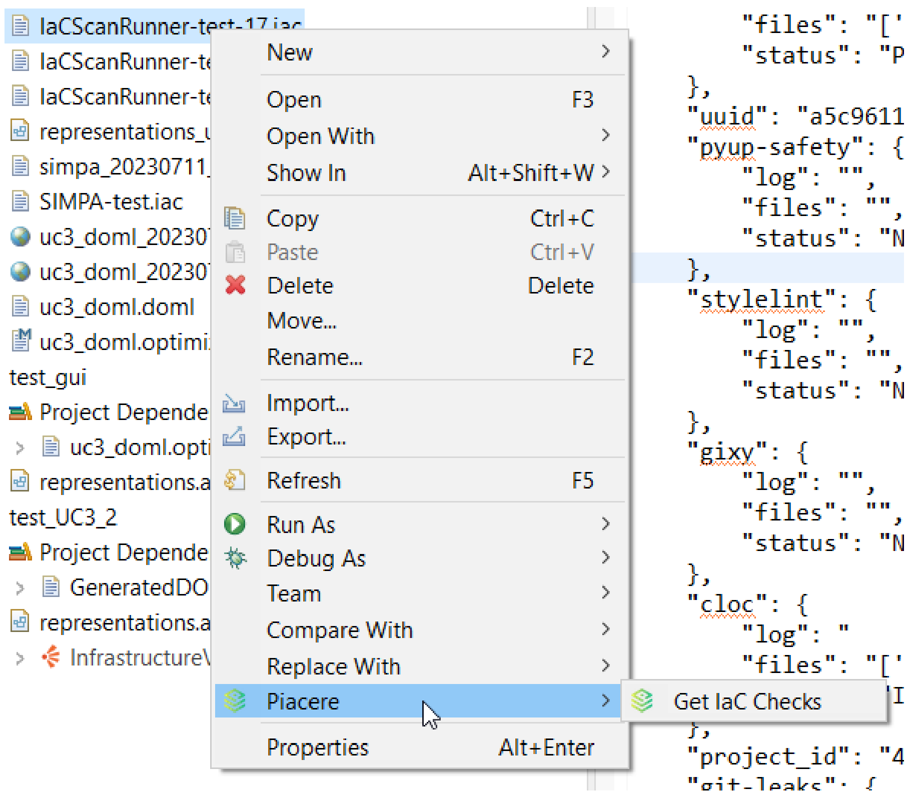
With the latest version of IaC Scan Runner supporting the Scan projects approach, the interface was brought to a level where user can manage multiple scan projects in the same workspace. This means that the user can organise workspace in a way to have an IaC scanning process for different stages, configure different list of checks per each project, and more. This also allows maintaining different sets of results and comparing them later.

To initiate the IaC Scan Runner, one has to create initiate Create IaC Project command (see Figure 21). The created project is represented with a file depicted with extension *.iac . The right click to an IaC file allows to get the list of IaC Scans that can be performed (Figure 22) and gives the form for changing the configuration (Figure 23). Afterwards one can initiate scan and select the IaC package (Figure 24). The last figure shows a short summary of the test (Figure 25).



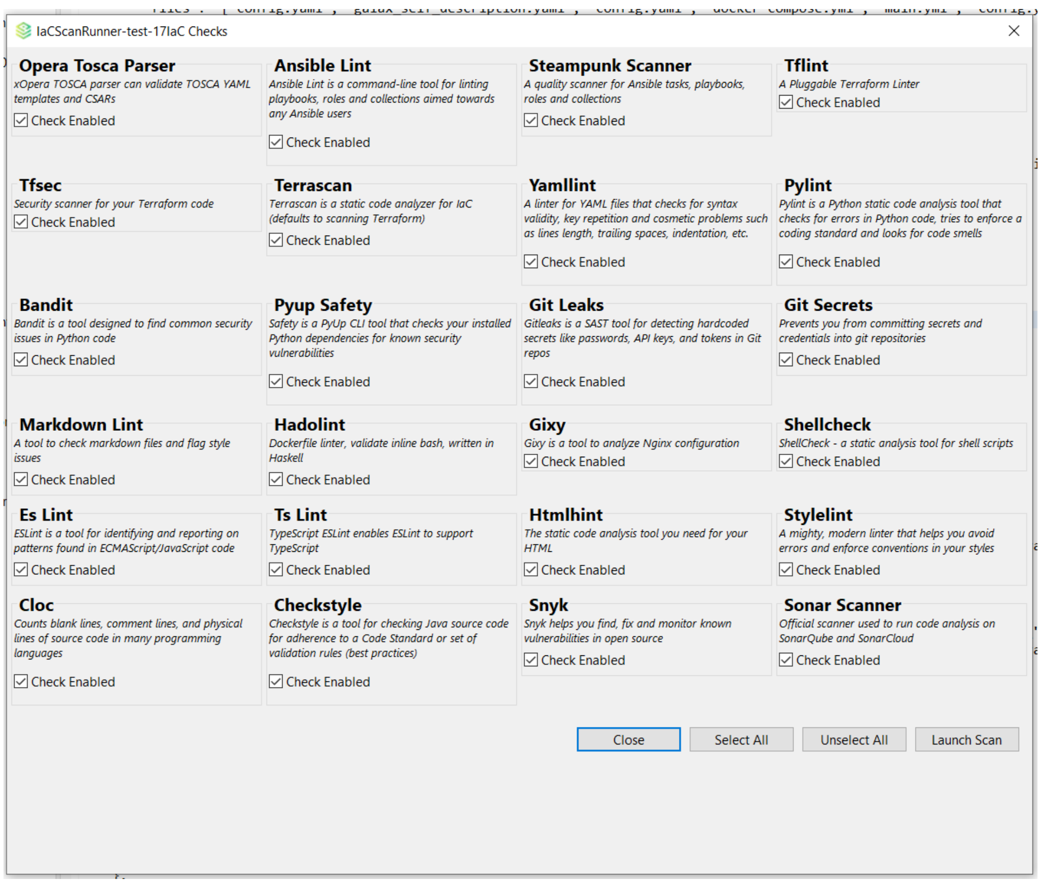
```
        name
        cidr
relation_
    depends_
        extend
    hosted_
        extend
    connect
        extend
    http+---
        extend
multi_id:
component_
    compute
        extend
        proper
            pub
                ty
            key_
                ty
            os_
                ty
            mach
                ty
            inst
                ty
    saas:
        extend
```

Figure 21 Create IaC Scan Project



```
    "files": ['
    "status": "P
},
"uuid": "a5c9611
"pyup-safety": {
    "log": "",
    "files": "",
    "status": "N
},
"stylelint": {
    "log": "",
    "files": "",
    "status": "N
},
"gixy": {
    "log": "",
    "files": "",
    "status": "N
},
"cloc": {
    "log": "
    "files": ['
                I
},
"project_id": "4
"git-leaks": {
```

Figure 22 Get the Scan Check list


IaCScanRunner-test-17IaC Checks

Opera Tosca Parser
xOpera TOSCA parser can validate TOSCA YAML templates and CSARs
☑ Check Enabled

Ansible Lint
Ansible Lint is a command-line tool for linting playbooks, roles and collections aimed towards any Ansible users
☑ Check Enabled

Steampunk Scanner
A quality scanner for Ansible tasks, playbooks, roles and collections
☑ Check Enabled

Tflint
A Pluggable Terraform Linter
☑ Check Enabled

Tfsec
Security scanner for your Terraform code
☑ Check Enabled

Terrascan
Terrascan is a static code analyzer for IaC (defaults to scanning Terraform)
☑ Check Enabled

Yamllint
A linter for YAML files that checks for syntax validity, key repetition and cosmetic problems such as lines length, trailing spaces, indentation, etc.
☑ Check Enabled

Pylint
Pylint is a Python static code analysis tool that checks for errors in Python code, tries to enforce a coding standard and looks for code smells
☑ Check Enabled

Bandit
Bandit is a tool designed to find common security issues in Python code
☑ Check Enabled

Pyup Safety
Safety is a PyUp CLI tool that checks your installed Python dependencies for known security vulnerabilities
☑ Check Enabled

Git Leaks
Gitleaks is a SAST tool for detecting hardcoded secrets like passwords, API keys, and tokens in Git repos
☑ Check Enabled

Git Secrets
Prevents you from committing secrets and credentials into git repositories
☑ Check Enabled

Markdown Lint
A tool to check markdown files and flag style issues
☑ Check Enabled

Hadolint
Dockerfile linter, validate inline bash, written in Haskell
☑ Check Enabled

Gixy
Gixy is a tool to analyze Nginx configuration
☑ Check Enabled

Shellcheck
ShellCheck - a static analysis tool for shell scripts
☑ Check Enabled

Es Lint
ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code
☑ Check Enabled

Ts Lint
TypeScript ESLint enables ESLint to support TypeScript
☑ Check Enabled

Htmlhint
The static code analysis tool you need for your HTML
☑ Check Enabled

Stylelint
A mighty, modern linter that helps you avoid errors and enforce conventions in your styles
☑ Check Enabled

Cloc
Counts blank lines, comment lines, and physical lines of source code in many programming languages
☑ Check Enabled

Checkstyle
Checkstyle is a tool for checking Java source code for adherence to a Code Standard or set of validation rules (best practices)
☑ Check Enabled

Snyk
Snyk helps you find, fix and monitor known vulnerabilities in open source
☑ Check Enabled

Sonar Scanner
Official scanner used to run code analysis on SonarQube and SonarCloud
☑ Check Enabled

Figure 23 Setting the Scan Check list

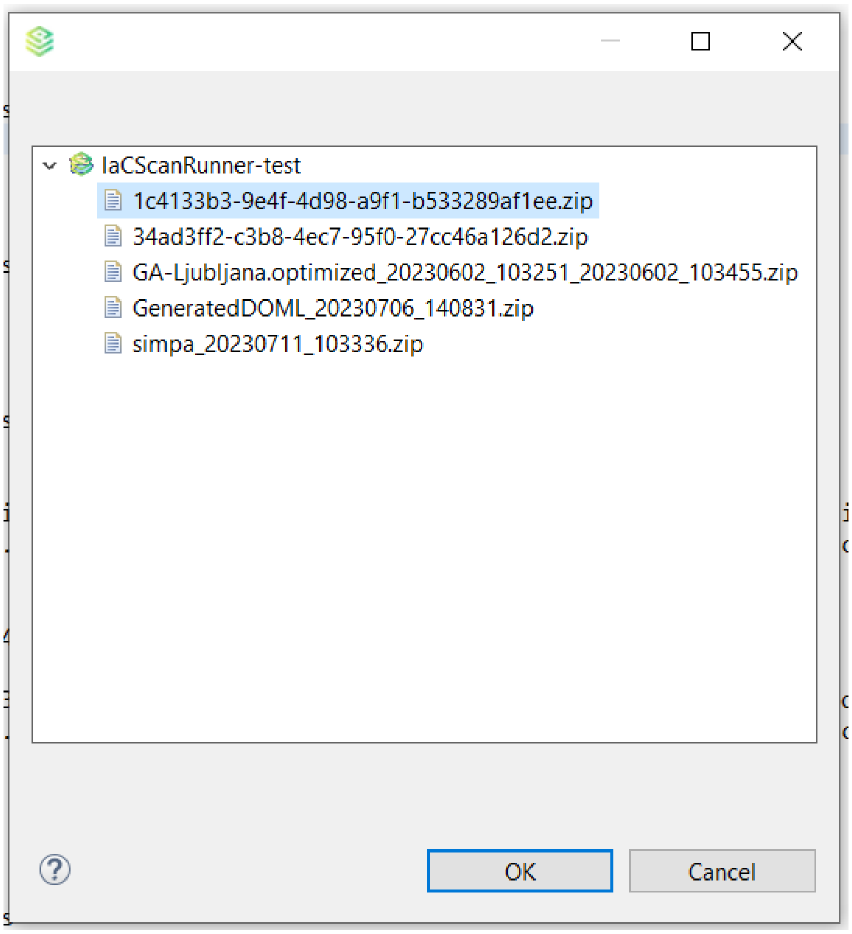
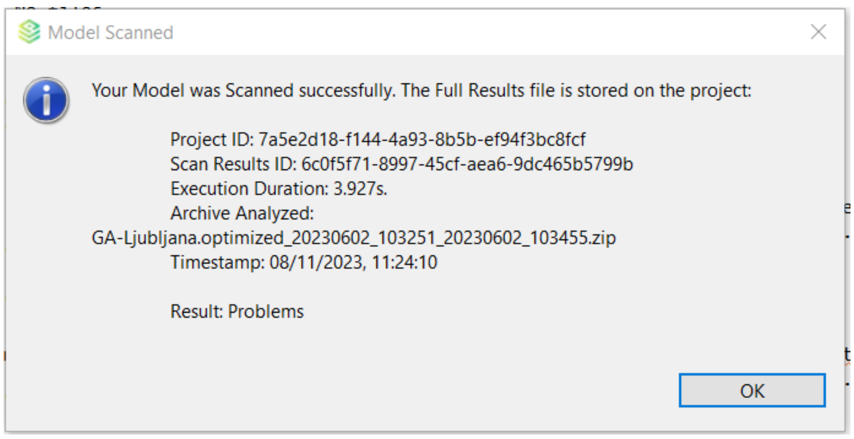

IaCScanRunner-test
1c4133b3-9e4f-4d98-a9f1-b533289af1ee.zip
34ad3ff2-c3b8-4ec7-95f0-27cc46a126d2.zip
GA-Ljubljana.optimized_20230602_103251_20230602_103455.zip
GeneratedDOML_20230706_140831.zip
simpa_20230711_103336.zip
OK
Cancel

Figure 24 The IaC package selection for scanning



Your Model was Scanned successfully. The Full Results file is stored on the project:

Project ID: 7a5e2d18-f144-4a93-8b5b-ef94f3bc8fcf
Scan Results ID: 6c0f5f71-8997-45cf-aea6-9dc465b5799b
Execution Duration: 3.927s.
Archive Analyzed:
GA-Ljubljana.optimized_20230602_103251_20230602_103455.zip
Timestamp: 08/11/2023, 11:24:10

Result: Problems

Figure 25 IaC Scan Result summary.

Inputs

Input to the IaC Scan Runner is a compressed package (zip or tar/compressed-tar), including the IaC code.

Outputs

Output is a text file (JSON or HTML) that includes the content – warnings and messages – that describe the issues in IaC and the return codes from each of the selected checks. In case of using IaC Scan Runner with project endpoints, the initial output is a short summary, while the detailed response is still available for the user to query from IaC Scan Runner service. In case of using the IDE, the output is also available in output file which appears in the Eclipse project workspace.

3.7 IDE – CSEP

The Canary Sandbox Environment Provisioner (CSEP) is a tool providing the PIACERE user with the ability to create (provision) Canary Sandbox Environments based on OpenStack.

The Canary Sandbox Environment Mocklord (CSEM) is a provider of mocked-up AWS APIs and is not meant for integration but as a target for, e.g., the PIACERE IEM.

Behaviour

It is expected that the PIACERE user will be able to issue new deployment requests and browse existing deployments from the IDE.

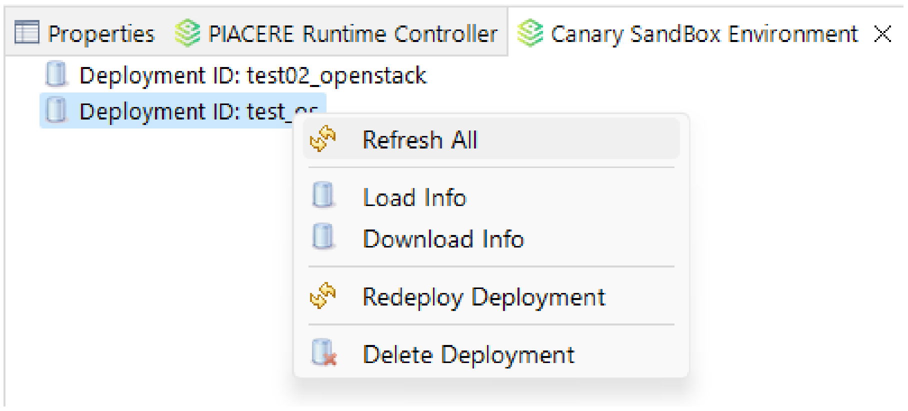
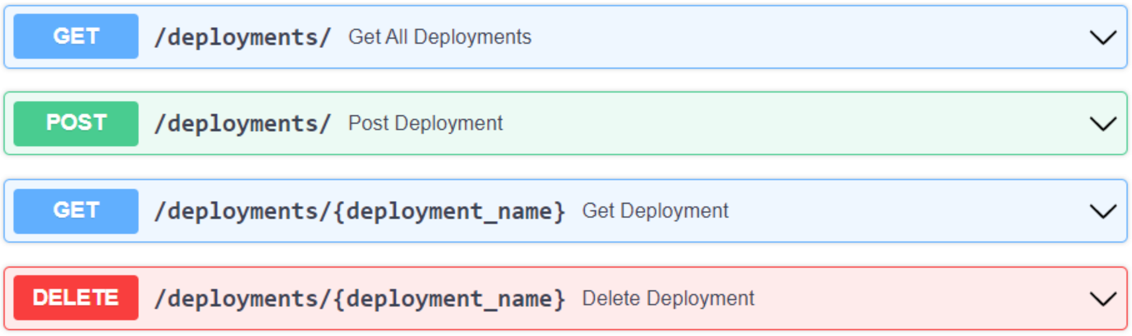


Figure 26 CSEP menu

The CSEP provides a REST API with endpoints required to facilitate this (Figure 27). The CSEP renders its own API documentation at the */docs* path which allows the integrator to inspect the API requirements.



GET /deployments/ Get All Deployments

POST /deployments/ Post Deployment

GET /deployments/{deployment_name} Get Deployment

DELETE /deployments/{deployment_name} Delete Deployment

Figure 27 CSEP API endpoints

There is a *dummy* deployment type provided for integrator's convenience which does not require actual target hosts to be available and the integration can be tested at no cost.

Inputs

As the input, the deployment creation API accepts JSON documents describing the details of the requested deployment. Each deployment is identified by its name which is a user-provided string and has to be unique on that instance of CSEP. The creation of deployments is facilitated by the *POST /deployments/* method. The other methods do not require any input.

Outputs

As the output, the API offers JSON documents describing the details of managed deployments.

3.8 IDE – PRC

The PIACERE Runtime Controller (PRC) is the orchestration tool of the PIACERE runtime. It is described in detail in this document in later sections.

Expected behaviour

The IDE uses PRC to control the runtime of PIACERE. The PRC allows to create new deployments and inspect the state of existing ones. The deployments are identified by their identifiers which are mostly free-form names. Please refer to PRC REST API section in this document for details on the behaviour and inputs and outputs.

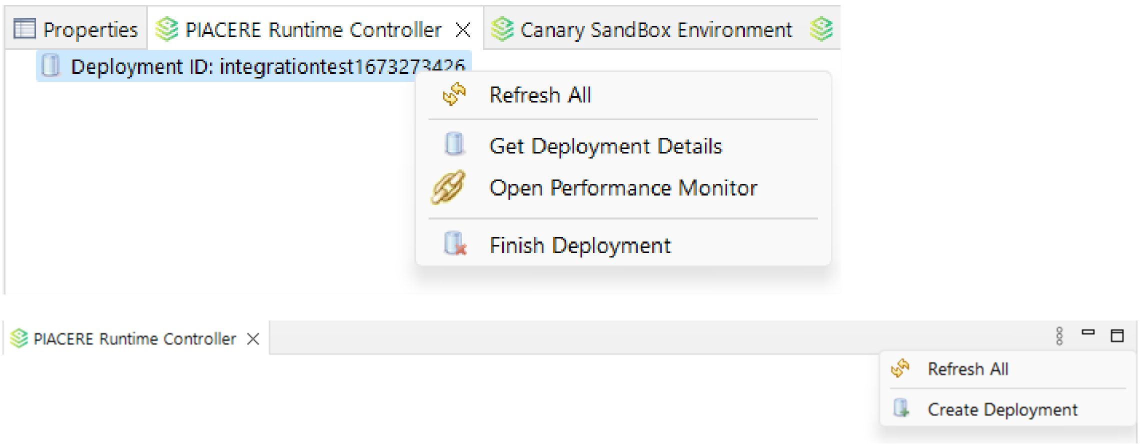

Properties
PIACERE Runtime Controller
Canary SandBox Environment
Deployment ID: integrationtest1673273426
Refresh All
Get Deployment Details
Open Performance Monitor
Finish Deployment
PIACERE Runtime Controller
Refresh All
Create Deployment

Figure 28 IDE – PRC menu and options

3.9 IDE – Monitoring Dashboard

The monitoring components allow gathering metric from the infrastructure resources that have been deployed by PIACERE framework. PIACERE supports two types of monitoring:

- 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. Examples of metrics are memory use, disk use, processes, CPU usage, etc.
- The **Security Monitoring** component focuses on gathering security-related measures from the infrastructure resources. The measures are gathered by agents running in the infrastructure resources.

Behaviour

When a new IaC is deployed, an extra task is undertaken: the deployment of monitoring agents along with the infrastructure, and the configuration of the monitoring components. Then, the continuous gathering of performance/security metrics can start. These metrics are evaluated against the expected thresholds or SLAs and, if needed, alerts are sent to the Self-Healing component.

The IDE enables the user to reach the dashboards of the monitoring system. In the next image we show the current approach to provide access on the monitoring aspects from the IDE. In the next versions we will also include access to the self-healing component to provide access to the monitoring event logs and self-healing strategies.

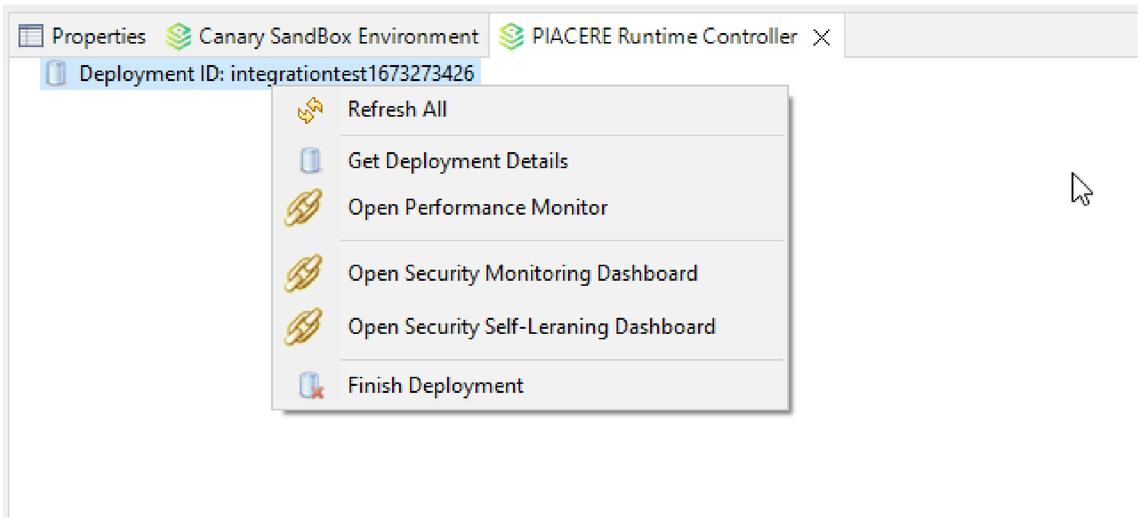
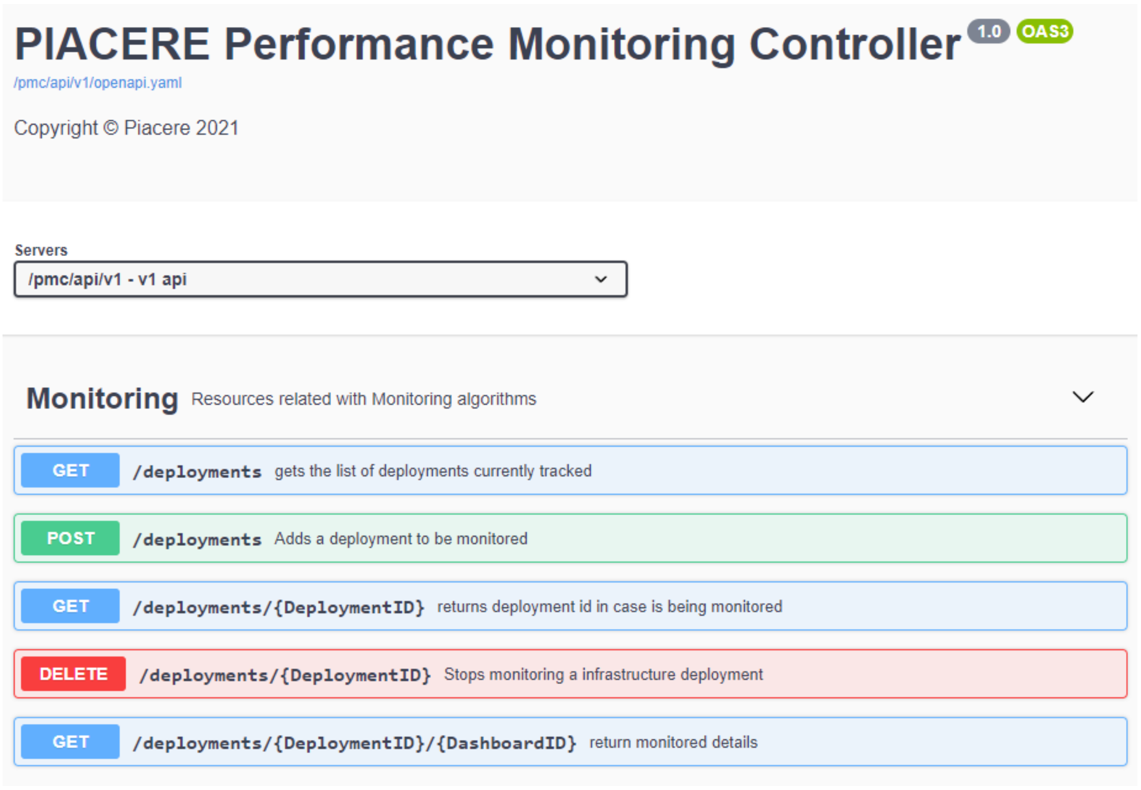

Properties
Canary SandBox Environment
PIACERE Runtime Controller
Deployment ID: integrationtest1673273426
Refresh All
Get Deployment Details
Open Performance Monitor
Open Security Monitoring Dashboard
Open Security Self-Leraning Dashboard
Finish Deployment

Figure 29 Monitoring access from deployment context menu

This is done via the Performance Monitoring Controller (PMC) which offers a REST API (Figure 30), the Security monitoring controller (Figure 31), and in the future the self-healing API (Figure 32).



PIACERE Performance Monitoring Controller 1.0 OAS3

/pmc/api/v1/openapi.yaml

Copyright © Piacere 2021

Servers
/pmc/api/v1 - v1 api

Monitoring Resources related with Monitoring algorithms

GET /deployments gets the list of deployments currently tracked

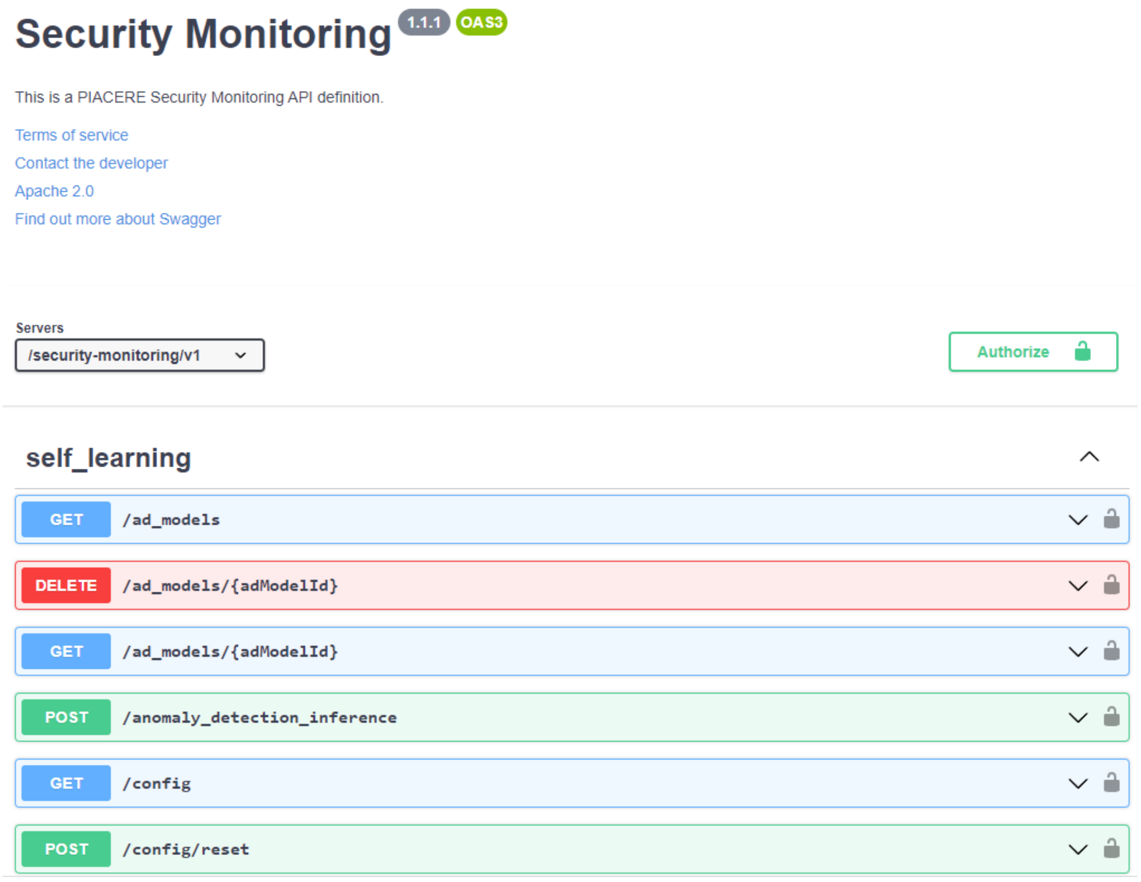
POST /deployments Adds a deployment to be monitored

GET /deployments/{DeploymentID} returns deployment id in case is being monitored

DELETE /deployments/{DeploymentID} Stops monitoring a infrastructure deployment

GET /deployments/{DeploymentID}/{DashboardID} return monitored details

Figure 30 Swagger UI page for PMC



Security Monitoring 1.1.1 OAS3

This is a PIACERE Security Monitoring API definition.

Terms of service
Contact the developer
Apache 2.0
Find out more about Swagger

Servers
/security-monitoring/v1

Authorize

self_learning

GET /ad_models

DELETE /ad_models/{adModelId}

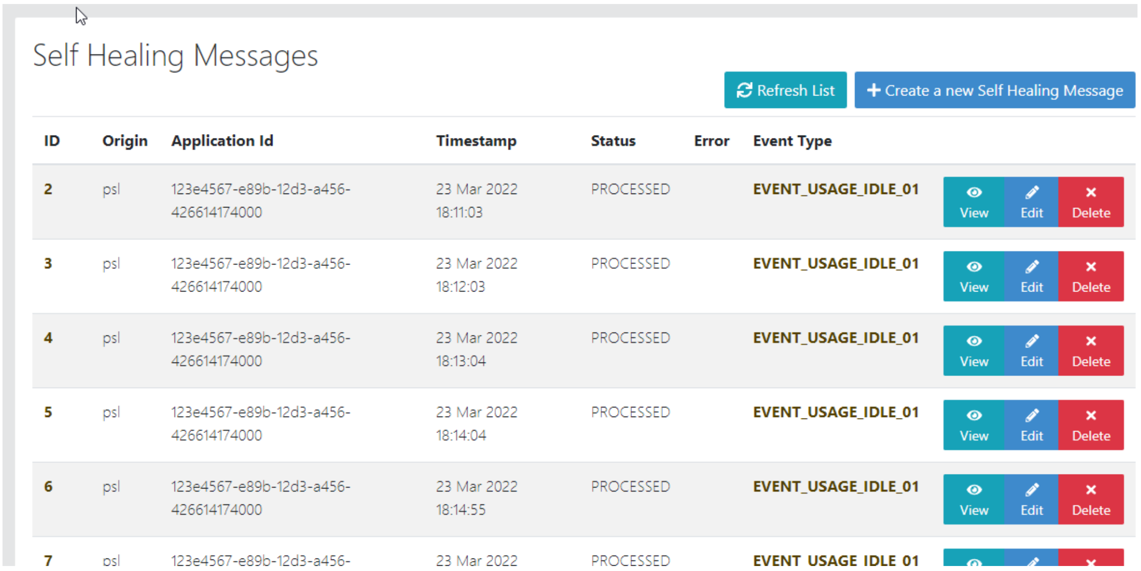
GET /ad_models/{adModelId}

POST /anomaly_detection_inference

GET /config

POST /config/reset

Figure 31 OpenApi for SMC



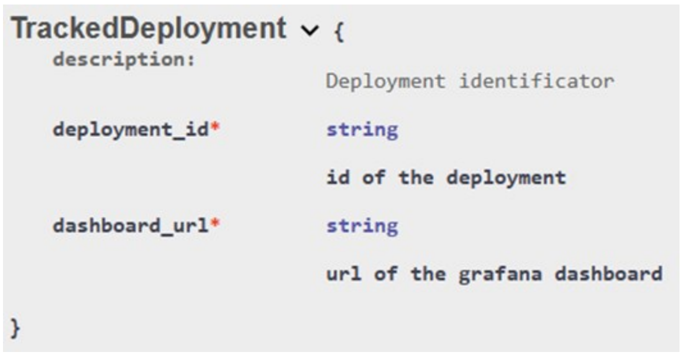
Self Healing Messages

ID	Origin	Application Id	Timestamp	Status	Error	Event Type	
2	psl	123e4567-e89b-12d3-a456-426614174000	23 Mar 2022 18:11:03	PROCESSED		EVENT_USAGE_IDLE_01	View Edit Delete
3	psl	123e4567-e89b-12d3-a456-426614174000	23 Mar 2022 18:12:03	PROCESSED		EVENT_USAGE_IDLE_01	View Edit Delete
4	psl	123e4567-e89b-12d3-a456-426614174000	23 Mar 2022 18:13:04	PROCESSED		EVENT_USAGE_IDLE_01	View Edit Delete
5	psl	123e4567-e89b-12d3-a456-426614174000	23 Mar 2022 18:14:04	PROCESSED		EVENT_USAGE_IDLE_01	View Edit Delete
6	psl	123e4567-e89b-12d3-a456-426614174000	23 Mar 2022 18:14:55	PROCESSED		EVENT_USAGE_IDLE_01	View Edit Delete
7	psl	123e4567-e89b-12d3-a456-	23 Mar 2022	PROCESSED		EVENT_USAGE_IDLE_01	

Figure 32 Event Logs in the SH

Inputs

In order to retrieve the URL of dashboard for a specific deployment, the performance monitoring controllers provide a method to get the URL of the dashboard. As an example of this approach, we show the method can be explored and tested from the Swagger UI embedded in the performance monitoring controller, and the response schema can be seen in Figure 33.



```
TrackedDeployment ⌄ {
   description:
                        Deployment identificator
   deployment_id*       string
                        id of the deployment
   dashboard_url*       string
                        url of the grafana dashboard
}
```

Figure 33 PMC – get deployment response

Outputs

The outputs to the IDE are the created Grafana dashboard's access link. The dashboard is still in design phase and the current draft is presented in Figure 34.

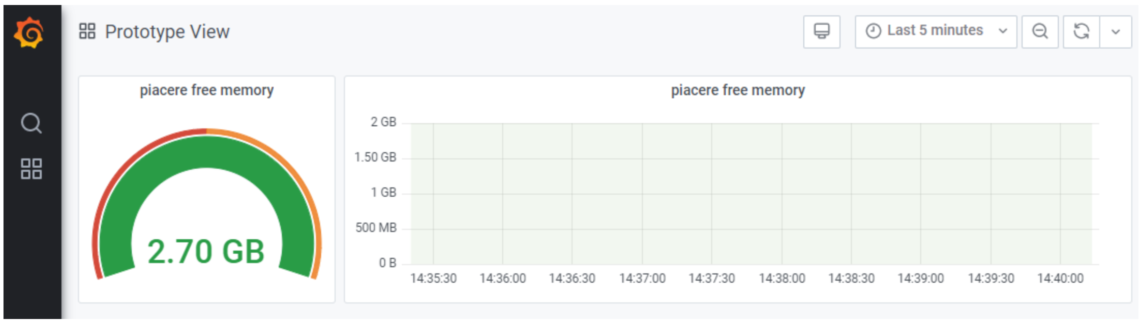
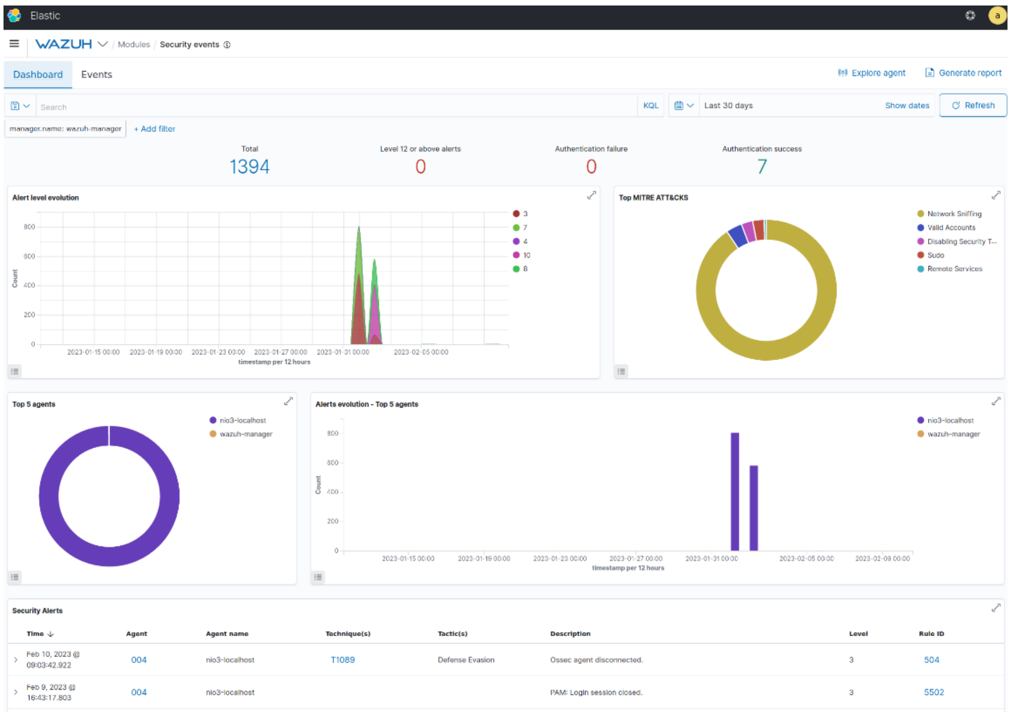


Figure 34 Grafana monitoring dashboard draft

The output of the Monitoring system is reflected in one or more visual dashboards, where the metrics, thresholds, etc. can be consulted in a graphical/numerical view. At the time of writing, the dashboards are not designed yet, but a graphic or table is expected for each of the metrics gathered like memory, disk, CPU usage or availability. These dashboards are created on the fly, once the information of the deployed infrastructure is available. For this, pre-defined templates or the Grafana API can be used. Grafana dashboards are HTML elements, and will be integrated in the PIACERE IDE, as iframes or a similar technology. Moreover, Security Monitoring Dashboard provides Kibana dashboard showing an overview of security-related events (Figure 35) throughout the time of the infrastructure's lifecycle (from the time of deployment of Security Monitoring agents on the infrastructure – which is implicitly already included with the definition of the rest of Monitoring infrastructure within DOML).



Security Alerts

Time	Agent	Agent name	Technique(s)	Tactic(s)	Description	Level	Rule ID
Feb 10, 2023 @ 09:03:42.922	004	nio3-localhost	T1089	Defense Evasion	Ossec agent disconnected.	3	504
Feb 9, 2023 @ 16:43:17.803	004	nio3-localhost			PAM: Login session closed.	3	5502

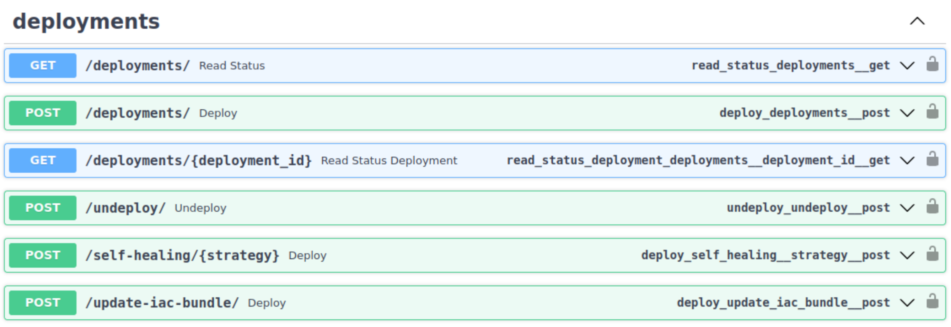
Figure 35 Security Monitoring Dashboard shows Wazuh's Kibana dashboard.

3.10 PRC – IEM

The IEM serves the purpose of deployment and redeployment of IaC. It supports different IaC technologies present in the market and provides a common interface to execute the deployments of the different elements present in the PIACERE project. In addition, the current prototype handles the redeployment and tearing down of existing deployments.

Behaviour

The main purpose of the IEM is to handle the lifecycle of the deployments that are managed within the PIACERE framework. The integration between the PRC and the IEM is handled by a REST API. The PRC can perform the following actions this way on the IEM: i) query the status of all active deployments, ii) trigger a deployment, iii) query the status of a particular deployment, iii) delete a deployment, iv) trigger a self-healing strategy, and v) update the iac bundle. These actions are depicted in Figure 36 as REST API endpoints.



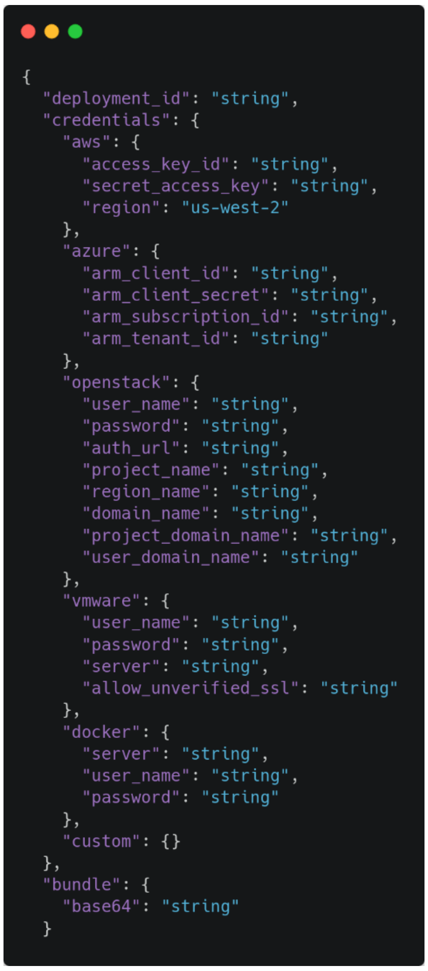
deployments

GET	/deployments/	Read Status	read_status_deployments__get
POST	/deployments/	Deploy	deploy_deployments__post
GET	/deployments/{deployment_id}	Read Status Deployment	read_status_deployment_deployments__deployment_id__get
POST	/undeploy/	Undeploy	undeploy_undeploy__post
POST	/self-healing/{strategy}	Deploy	deploy_self_healing__strategy__post
POST	/update-iac-bundle/	Deploy	deploy_update_iac_bundle__post

Figure 36 IEM Endpoints

Inputs

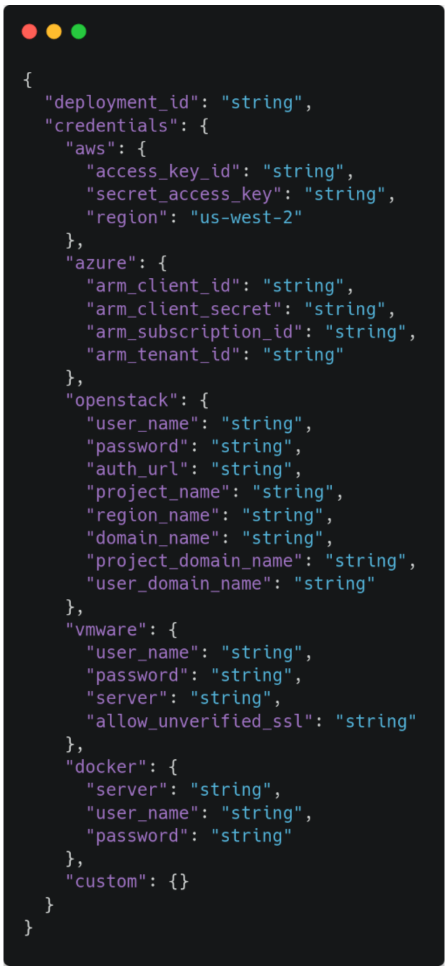
The POST */deployments/* triggers or updates a deployment in the infrastructure. The IEM needs a few fields to kick off a deployment, the schema for calling this endpoint is showcased in Figure 37. This schema is also valid for the POST /update-iac-bundle endpoint that oversees the redeployment of the ecosystem via the self healing strategy.



```
{
  "deployment_id": "string",
  "credentials": {
    "aws": {
      "access_key_id": "string",
      "secret_access_key": "string",
      "region": "us-west-2"
    },
    "azure": {
      "arm_client_id": "string",
      "arm_client_secret": "string",
      "arm_subscription_id": "string",
      "arm_tenant_id": "string"
    },
    "openstack": {
      "user_name": "string",
      "password": "string",
      "auth_url": "string",
      "project_name": "string",
      "region_name": "string",
      "domain_name": "string",
      "project_domain_name": "string",
      "user_domain_name": "string"
    },
    "vmware": {
      "user_name": "string",
      "password": "string",
      "server": "string",
      "allow_unverified_ssl": "string"
    },
    "docker": {
      "server": "string",
      "user_name": "string",
      "password": "string"
    },
    "custom": {}
  },
  "bundle": {
    "base64": "string"
  }
```

Figure 37 IEM PUT /deployments/ request schema.

The POST /undeploy/ serves the purpose of tearing down a particular deployment. In Figure 38, the request body required for the invocation of this endpoint is shown. This is also valid for the POST /self-healing/{strategy}

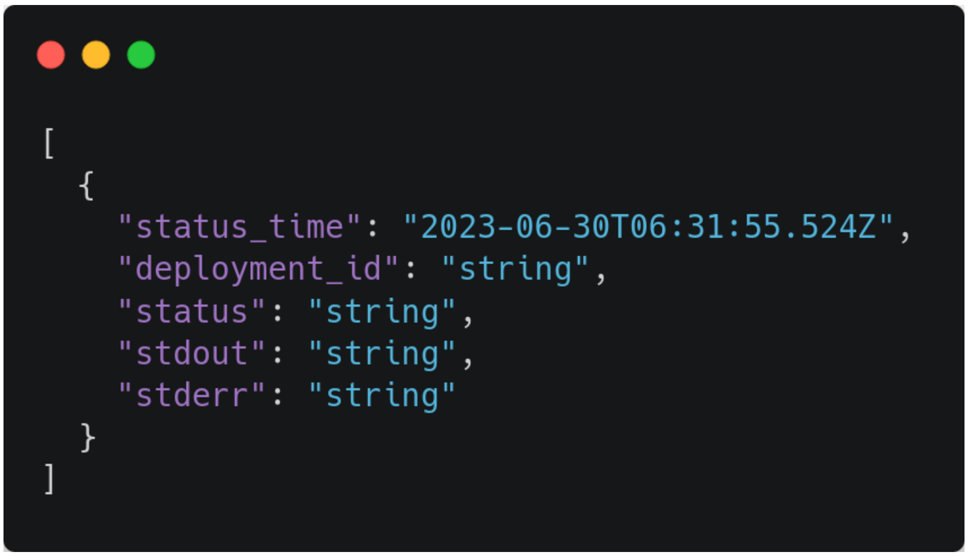


```
{
  "deployment_id": "string",
  "credentials": {
    "aws": {
      "access_key_id": "string",
      "secret_access_key": "string",
      "region": "us-west-2"
    },
    "azure": {
      "arm_client_id": "string",
      "arm_client_secret": "string",
      "arm_subscription_id": "string",
      "arm_tenant_id": "string"
    },
    "openstack": {
      "user_name": "string",
      "password": "string",
      "auth_url": "string",
      "project_name": "string",
      "region_name": "string",
      "domain_name": "string",
      "project_domain_name": "string",
      "user_domain_name": "string"
    },
    "vmware": {
      "user_name": "string",
      "password": "string",
      "server": "string",
      "allow_unverified_ssl": "string"
    },
    "docker": {
      "server": "string",
      "user_name": "string",
      "password": "string"
    },
    "custom": {}
  }
}
```

Figure 38 IEM DELETE /deployments/ request schema

Outputs

The output is present for GET */deployments/{deployment_id}* and its schema is shown in Figure 39. This is also valid for the GET /deployments/ endpoint, that oversees querying for all the deployments.



```
[
  {
    "status_time": "2023-06-30T06:31:55.524Z",
    "deployment_id": "string",
    "status": "string",
    "stdout": "string",
    "stderr": "string"
  }
]
```

Figure 39 IEM GET /deployments/{deployment_id} response schema.

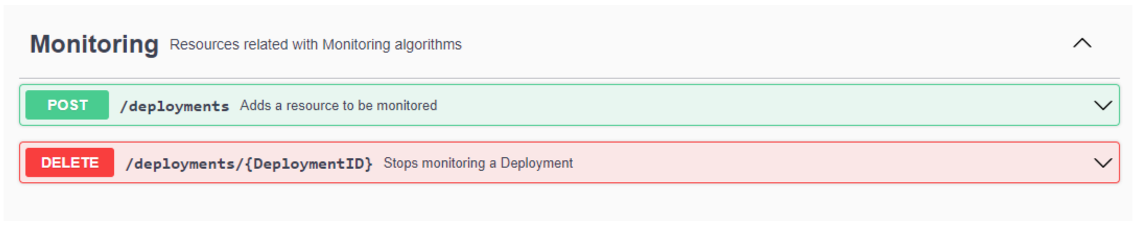
3.11 PRC – Monitoring Controller

The monitoring controller (MC) is a utility component that aims to simplify the configuration of the monitoring, self-learning and self-healing component as PIACERE platform creates, updates and destroys deployments. It also isolates the PRC from changes in those mentioned components.

Behaviour

This component is in charge of the activation and deactivation of monitoring activities throughout all the monitoring components: performance monitoring, security monitoring, performance Self-learning and security Self-learning. It exposes a REST API (Figure 40).

PRC calls this component when initiating a new deployment to configure the monitoring stack for it.



Monitoring Resources related with Monitoring algorithms

POST /deployments Adds a resource to be monitored

DELETE /deployments/{DeploymentID} Stops monitoring a Deployment

Figure 40 MC OpenAPI rendered in embedded Swagger UI

Inputs

The input is the deployment identifier of the deployment to be monitored as well as the IaC bundle.

Outputs

The only output is an acknowledge that the request has been received and it is being processed (that is, starting up monitoring and self-learning components).

3.12 Self-Healing – PRC

The PIACERE Runtime Controller (PRC) is the orchestration tool of the PIACERE runtime. It is described in detail in this document in later sections.

Expected behaviour

The Self-Healing uses PRC to issue authenticated IEM command required for its self-healing strategies. The PRC allows this via a dedicated per-deployment endpoint. Each deployment is identified by an identifier which is a mostly free-form name. Please refer to PRC REST API section in this document for details on the behaviour and inputs and outputs.

3.13 Monitoring Component – IEC

The purpose of this interaction is to add and update real time information about the quality (i.e., a set of attributes) pertaining to the infrastructure element.

Behaviour

The Monitoring gets real-time data from the different infrastructural elements used during the operation of the started deployments. This information will be sent back to the IEC to its future use in IOP to adjust the properties that are used during the filtering and selection of the infrastructure elements.

Monitoring will use the IEC REST API to keep some attributes up to date for each of the infrastructure elements in use. It will add and update information about the number of incidences with respect to performance, availability, and security.

Inputs

The information to be provided for IEC is a reference of the IEC element affected and the type of the value that is being updated and the actual value to be included.

Outputs

The output is the confirmation that the request has been processed and resources updated.

4 PRC Implementation

4.1 Changes since v2

The internals of PRC have changed substantially between v2 and v3. Some follow the future plans sketched in the previous version of this deliverable, others are a result of the new approach to self-healing.

4.2 Functional description

The PRC (PIACERE Runtime Controller) is the heart of the PIACERE runtime integration. It is the main component of the runtime called from the design-time tooling (IDE). It is also the main component coordinating the work of the runtime, i.e., calling the other specific tools delivered as part of the PIACERE framework for the runtime and allowing those tools to call back.

The runtime of PIACERE is responsible for ensuring that the designed infrastructure is deployed, monitored, self-healed, and optimized as expected. To this end, the basic concept in the runtime is deployment. Each deployment is managed individually and can be updated by providing an updated design (specifically, IaC). In v3, the IaC is provided directly to the PRC, without the Git repository intermediary to make it more decoupled.

There were no PRC-specific requirements collected. The PRC satisfies the self-imposed requirement of being usable to realise the designed workflow.

The main innovation of PRC is the integration of runtime components to create a coherent workflow for deployment, its monitoring, self-healing and optimization.

4.2.1 Fitting into the overall PIACERE Architecture

The PRC coordinates the workflow, the other tools themselves are concerned with work details. The PRC is responsible for calling the other tools at relevant times. To this end, it has to communicate with them using the interfaces they offer.

There are 2 tools that are called by the PRC v3:

- IEM (IaC Execution Manager): responsible for executing IaC, called due to creation of a new deployment, update of an existing deployment or in the course of a relevant self-healing strategy,
- MC (Monitoring Controller): responsible for configuring the WP6 monitoring, self-learning and self-healing suite of tools.

PRC v3 itself is called by 2 PIACERE tools:

- IDE (the PIACERE GUI in Eclipse): the IDE is user's default starting point for interaction with PRC, the initial deployment request will likely originate from it,
- Self-Healing (SH): SH uses PRC to request authenticated IEM actions on its behalf.

4.3 Technical description

4.3.1 Components description

The PRC v3 is made of a single component, the API, which offers the PRC REST API (HTTP) interface that can be used by the external world. This is to be used by other PIACERE tools, such as the IDE and Self-Healing. The interface itself is described in detail in the User Manual section. The API component handles all the communication aspects internally and no longer relies on intermediaries to make it simpler, more flexible and focused strictly on its role of being the runtime intermediary, handling the authentication for IEM requests and ensuring that the monitoring subsystem is informed about the new deployment.

4.3.2 Technical specifications

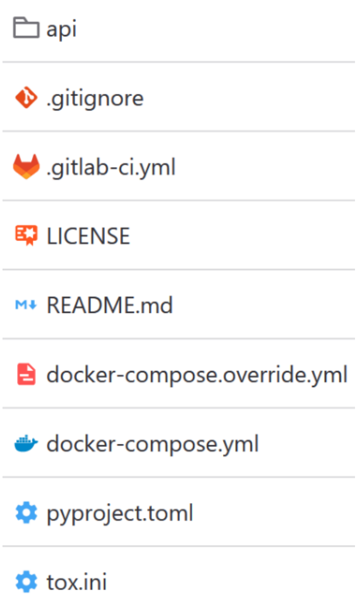
The PRC API is written in Python and tested with Python 3.10. The PRC API uses the FastAPI [5] framework, currently in version 0.95.0. This framework supports Asynchronous Server Gateway Interface (ASGI). ASGI must be understood by the used web server. The one chosen for this project is uvicorn [6], currently in version 0.21.1. FastAPI relies on Python's type annotations to handle API definition and validation of inputs and outputs. FastAPI also generates OpenAPI specification automatically and allows rendering docs in Swagger UI [7] and Redoc [8]. Additionally, Pydantic [9] is used both indirectly, for the purposes of FastAPI, as well as directly for the management of settings (via environment variables). Pydantic's version is managed by FastAPI. Pydantic provides additional type annotations and validation of objects. Finally, a suite of robust, asynchronous libraries for networking (aiodns, aiohttp) is used to handle the communication with other components.

5 PRC Delivery and usage

5.1 Changes since v2

The PRC's delivery and usage have both changed considerably since v2. These changes reflect the re-architecture of PIACERE runtime described in the previous sections.

5.2 Package information



api
.gitignore
.gitlab-ci.yml
LICENSE
README.md
docker-compose.override.yml
docker-compose.yml
pyproject.toml
tox.ini

Figure 41 PRC package contents

The contents of the package at-a-glance are shown in Figure 41 (on the left). The most important elements are listed and described below:

- **api** directory contains a Python module with the PRC API layer as well as the Dockerfile and requirements listing necessary to build reproducible container images,
- **docker-compose.yml** and **docker-compose.override.yml** together offer a quick way to deploy PRC on any Docker-enabled machine,
- **pyproject.toml** and **tox.ini** define the code quality checks and applicable linting rules.

5.3 Installation instructions

The software is offered containerised and the container image recipe is provided along with an example docker-compose deployment. The users are recommended to install a relatively modern Docker (19.03+) and docker-compose (1.29+).
Then, it is only a matter of running *docker-compose up -d* to get the software running.

5.4 User Manual

The PRC offers interfaces to allow for integration. These are dedicated to the tools expected to be able to integrate with PRC.

The main assumption is that the user is able to create a process instance which overlooks a particular deployment, allowing to trigger further actions on it. The deployment (process instance) should be identified by a name provided by the user (there could be an additional identifier, but the name has to be easily usable for integration purposes as it is passed in by the tools and passed on to the tools).

IDE actions:

- Creating a deployment
- Getting deployments
- Get details (status) of a deployment
- Update an existing deployment

Self-healing actions:

- Update an existing deployment

5.4.1 PRC REST API

This section contains a detailed description of PRC REST API. The API revolves around deployments, each identified by unique user-provide **deployment_id**.

Below, a list of all exposed endpoints is presented.

- Create a new deployment:

POST /deployments

With a request body:

```
{
  "deployment_id": "string",
  "deployment_bundle": {
    "base64": "string"
  },
  "credentials": {
    ...
  }
}
```

The IaC bundle is provided in the request, base64-encoded. The credentials are opaque to PRC and are stored to be passed safely to IEM on each occasion.

- Get a list of all deployments:

GET /deployments

The response is a list (potentially empty) of all deployments.

- Get a specific deployment:

GET /deployments/{deployment_id}

Possible responses are a deployment description with status code 200 or a resource not found as an error with status code 404.

- Replay (aka "redeploy") the deployment

POST /deployments/{deployment_id}/redeploy

This action causes the deployment to be realised again, possibly using a modified IaC bundle and/or credentials.

- Undo (aka "undeploy") the deployment

POST /deployments/{deployment_id}/undeploy

This action causes the deployment to be undone.

5.5 Licensing information

The PRC is closed-source software, licensed for the PIACERE consortium.

5.6 Download

The code is available for review at Tecnalia's GitLab, in the private part of the PIACERE project: https://git.code.tecnalia.com/piacere/private/t23-runtime-controller

The source code can be provided under request through an email to the address appearing on the website (https://www.piacere-project.eu/) in the footer under "Contact Us". However, no open-source license is granted.

6 Conclusions

This deliverable has described the integration of the PIACERE framework (KR13) v3 and the crucial component of it – the PRC – in detail. The context, architecture, implementation, and usage of PRC were all described.

This final integration involves all tools delivered in year 3 (after M30) of the PIACERE project, that is the KRs of other technical WPs: WP3-WP6. The tools used in the integration were summarised in this document and important integration points described.

7 References

[1] ‘The OpenAPI Specification’. OpenAPI Initiative, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/OAI/OpenAPI-Specification

[2] ‘The Moby Project (Docker)’. Moby, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/moby/moby

[3] ‘Docker Compose’. Docker, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/docker/compose

[4] ‘traefik/traefik’. Traefik Labs, Feb. 18, 2022. Accessed: Feb. 18, 2022. [Online]. Available: https://github.com/traefik/traefik

[5] S. Ramírez, ‘tiangolo/fastapi’. Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/tiangolo/fastapi

[6] ‘encode/uvicorn’. Encode, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/encode/uvicorn

[7] ‘swagger-api/swagger-ui’. Swagger, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/swagger-api/swagger-ui

[8] ‘redoc’. Redocly, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/Redocly/redoc

[9] S. Colvin, ‘pydantic’. Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/samuelcolvin/pydantic