



PIACERE

Deliverable D5.2

IaC Execution Manager Prototype – v2

Editor(s):	Josu Díaz de Arcaya
Responsible Partner:	Tecnia
Status-Version:	v1.0
Date:	30.11.2022
Distribution level (CO, PU):	PU

Project Number:	101000162
Project Title:	PIACERE

Title of Deliverable:	IaC Execution Manager Prototype
Due Date of Delivery to the EC	30.11.2022

Workpackage responsible for the Deliverable:	WP5 - Package, release and configure IaC
Editor(s):	Josu Díaz de Arcaya (Tecnalia Research & Innovation)
Contributor(s):	Josu Díaz de Arcaya (Tecnalia Research & Innovation)
Reviewer(s):	Giuseppe Celozzi (Ericsson)
Approved by:	All partners
Recommended/mandatory readers:	WP2, WP3, WP5, WP6,WP7/ WP8

Abstract:	The main outcomes of Task 5.2 - PIACERE IaC Execution Manager Prototype are presented in this deliverable. This deliverable corresponds to Key Result 10.
Keyword List:	IaC Execution Manager
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	13.10.2022	First draft version	Tecnia
v0.2	25.11.2022	Review	Ericsson
v0.3	28.11.2022	Address Comments	Tecnia
v1.0	30.11.2022	Ready for submission	Tecnia

DRAFT

Table of contents

Terms and abbreviations.....	6
Executive Summary.....	7
1 Introduction	8
1.1 About this deliverable	8
1.2 Document structure.....	8
2 Tools aiding in the operationalization life cycle.....	9
2.1 Current state of art.....	9
2.1.1 Orchestration in H2020 projects.....	9
2.2 Analysis of shortcomings.....	9
2.2.1 Applicability in the current project	10
2.3 PIACERE IaC Execution Manager goals.....	11
2.4 PIACERE IaC Execution Manager’s relevance to use cases	11
3 Implementation.....	12
3.1 Changes in v2	12
3.2 Requirements covered by this prototype	13
3.3 Functional description.....	14
3.3.1 Fitting into overall PIACERE Architecture.....	15
3.4 Technical description	16
3.4.1 Prototype architecture.....	16
3.4.2 Component description.....	17
3.4.3 Component interactions	17
3.4.4 Technical specifications.....	19
3.4.5 Project skeleton.....	20
3.4.6 Public and private cloud provisioners	21
3.4.7 Challenges and pitfalls.....	22
4 Delivery and usage	24
4.1 Package information	24
4.2 Installation instructions.....	25
4.3 User Manual	29
4.4 Licensing information.....	31
4.5 Download	31
5 Conclusions	32
6 References.....	33
Appendix	35

List of tables

TABLE 1 - REQUIREMENTS THAT NEED TO BE ADDRESSED BY THE IEM AND ITS CURRENT FULFILMENT STATUS..	13
---	----

TABLE 2 - EXCERPT SHOWING THE BUILT AND EXECUTION OF THE IEM CONTAINER	28
TABLE 3 - EXCERPT SHOWING THE RELEVANT URL'S OF THE IEM	31

List of figures

FIGURE 1 – PIACERE RUNTIME DEPLOY DIAGRAM (VERSION 2.1)	15
FIGURE 2 - PIACERE RUNTIME MONITORING (VERSION 2.1)	16
FIGURE 3 - ARCHITECTURE OF THE DIFFERENT ELEMENTS COMPRISING THE IEM EXECUTION MANAGER	16
FIGURE 4 - SEQUENCE DIAGRAM FOR THE PROCESS OF KICKING OFF A DEPLOYMENT	17
FIGURE 5 – SEQUENCE DIAGRAM OF THE PROCESS OF REQUESTING THE STATUS OF A GIVEN DEPLOYMENT	18
FIGURE 6 - SEQUENCE DIAGRAM FOR THE PROCESS OF TEARING DOWN A DEPLOYMENT	19
FIGURE 7 - SAMPLE PROJECT WITH VARIOUS SEQUENTIAL STAGES.	20
FIGURE 8 - CONFIGURATION FILE STATING THE SEQUENTIAL EXECUTION ORDER OF THE DEPLOYMENT.	20
FIGURE 9 - NGINX STAGE FOLDER STRUCTURE.	21
FIGURE 10 - CONFIGURATION INFORMATION FOR THE NGINX STAGE.	21
FIGURE 11 - CREDENTIALS SECTION THAT ARE FED INTO THE IEM TO KICK OFF A DEPLOYMENT.....	22
FIGURE 12 – ROOT FOLDER FOR PIACERE'S IEM COMPONENT	24
FIGURE 13 – CONTINUOUS INTEGRATION AND DEPLOYMENT PIPELINE FOR THE IEM PROJECT	24
FIGURE 14 – SONARQUBE DASHBOARD OF THE IEM COMPONENT.....	25
FIGURE 15 - EXCERPT SHOWING THE INSTALLATION STEPS FOR VIRTUALENV IN A LINUX BOX	26
FIGURE 16 - EXCERPT SHOWING THE CREATION OF A VIRTUAL ENVIRONMENT	26
FIGURE 17 - EXCERPT SHOWING THE ACTIVATION OF THE FRESHLY CREATED VIRTUAL ENVIRONMENT	26
FIGURE 18 - EXCERPT SHOWING HOW TO FINALIZE OR DEACTIVATE A VIRTUAL ENVIRONMENT	26
FIGURE 19 - EXCERPT SHOWING THE CONTENT OF THE REQUIREMENTS.TXT FILE AND HOW TO INSTALL THEM..	27
FIGURE 20 - EXCERPT SHOWING A SUCCESSFUL EXECUTION OF THE TEST OF THE IEM.....	27
FIGURE 21 - EXCERPT SHOWING HOW TO KICK OFF THE IEM.....	28
FIGURE 22 - EXCERPT SHOWING THE DOCKERFILE THAT WILL BE USED TO GENERATE THE CONTAINERIZED IMAGE OF THE IEM	28
FIGURE 23 - OPENAPI SPECIFICATION FOR THE INTERACTION WITH THE IEM COMPONENT	29
FIGURE 24 - REQUEST BODY FOR THE PUT /DEPLOYMENTS/ ENDPOINT.....	29
FIGURE 25 - EXCERPT SHOWING THE MAIN.PY FILE IN WHICH THE DIFFERENT ENDPOINTS FOR THE IEM RESIDE	30
FIGURE 26 - EXCERPT SHOWING THE CLASS HOLDING THE MODELS USED FOR THE COMMUNICATION WITH THE REST API.....	30
FIGURE 27 - EXCERPT SHOWING HOW TO INITIALIZE THE IEM WITH THE UVICORN SERVER	31

Terms and abbreviations

CE	Canary Environment
CLI	Command Line Interface
CSE	Canary Sandbox Environment: term used in DoA instead of CE
CSP	Cloud Service Provider
DevOps	Development and Operations
DevSecOps	Development, Security and Operations
DoA	Description of Action
EC	European Commission
GA	Grant Agreement to the project
HW	Hardware
IaC	Infrastructure as Code
IEP	IaC Execution Platform
IOP	IaC Optimizer
KPI	Key Performance Indicator
KR	Key Result
SW	Software
SaaS	Software as a Service
IEM	IaC Execution Manager
IaC	Infrastructure as Code

DRAFT

Executive Summary

This document contains the technical description of the IaC Execution Manager prototype (IEM) that is being developed within the context of work package 5 - Package, release and configure Infrastructure as Code - of the PIACERE project. The IEM is the central piece that handles the execution of the IaC code being generated in the early stages of the PIACERE framework. 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.

A state of the art of the different technologies that can be used for the provisioning, configuration, and orchestration of the different infrastructural devices that can be found in a production deployment is offered. This has served us to provide evidence and reasoning for the selection of the technologies that the IEM prototype is going to utilize. In addition, the PIACERE requirements for the IEM are presented, including which ones are currently covered and which ones will be covered in future releases of this prototype. Regarding these requirements, 66% of them are already partially implemented. Next, the software component is explained, sequence diagrams are provided for a better understanding, and the installation process and how to use it is also clarified in depth.

At its current stage, this prototype is viable for the deployment of different IaC technologies that cover the provisioning and the configuration of the infrastructural devices required for the projects utilizing the PIACERE framework. It provides a unified interface for other components so they can interact with the IEM in a unified manner. It can also be deployed in production utilizing container-based technologies which makes this prototype viable to be operationalized in public and private cloud provides, and on premises. For this prototype, the IEM supports two well established technologies (i.e., Ansible and Terraform) that are able to provision the different infrastructural devices required by the use cases, and the configuration of each of these infrastructural devices so they can accommodate the applications to be allocated. In addition, the interaction with this component has been hardened with authentication technologies so that its services cannot be misused by malicious third parties. Next, a persistence layer implemented to provide other components with metrics and data regarding past and present deployments. Finally, integration efforts have been conducted with other PIACERE components to interact successfully.

As for the future steps, some other technologies regarding the provisioning, configuration, and deployment orchestration might be included in the IEM, if they are required by the different use cases. Finally, further integration efforts will be conducted to make sure its operation and integration with the rest of the PIACERE ecosystem operates seamlessly. In addition, the IEM is going to be able to trigger specific strategies that will act on an existing deployment.

1 Introduction

This section offers a summary about the content of this document and showcases the document structure in detail.

1.1 About this deliverable

This document revolves around the IEM prototype which is part of the wider PIACERE framework. The IEM serves the purpose of deployment and redeployment of the different scenarios that are developed by the PIACERE use cases. Being at month M24 of the project, the prototype described here is the second version, that is going to be integrated in the PIACERE integrated framework version 2 at M27. In this document, we elaborate on the different technologies and alternatives that can be used for the development of this prototype. The purpose of this software deliverable is addressing PIACERE's KR10. In addition, detailed information on the software component is provided alongside its functionalities. Finally, future goals and conclusions are presented.

1.2 Document structure

This document is an extension of the one delivered on the first year of the project (D5.1 [1]). The parts that have not been modified during this second year are referenced to the proper deliverable. However, updated content and relevant sections have been kept and addressed in this same document for completeness. The document is structured in the following way.

Section 2 contains a description of the different technologies in the industry that can be utilized to implement Infrastructure as Code technologies, it also showcases the technologies utilized by the use cases and the relevance of this IEM prototype for the different use cases. The complete state of the art is referenced to the previous year deliverable, and only new material in this regard has been kept. Section 3 revolves around the current implementation of the IEM prototype. In it, the requirements currently covered by this prototype are shown. Requirements that will be covered during future developments of this project are also showcased and marked as such and have been updated. In addition, a functional and technical description is offered for a better understanding of the prototype. The existing architecture diagrams have been updated from last year, and a fresh new one for the runtime monitoring, and the role of the IEM in this field is included. Section 4 offers detailed information about the installation of the prototype and how to use it from scratch. Then, the conclusions are presented in Section 5. Finally, Section 6 showcases the features of the IEM in the Gherkin format, which promotes the usability of this component within the project.

2 Tools aiding in the operationalization life cycle

2.1 Current state of art

In [1] the complete state of the art of relevant technologies that have been assessed in the development of this component can be found. In addition, the applicability of these tools in the context of the PIACERE project alongside their shortcomings is offered.

An interesting technology which may become handy for the purpose of the project is Packer by HashiCorp [2], which promotes the creation and management of machine images for the different public and private cloud providers from source. Given the fact that one of the scenarios of the IEM is the deployment of projects in different providers Packer might reduce the differences between them, hence minimizing the configuration problem caused by image heterogeneity.

2.1.1 Orchestration in H2020 projects

There are a few Horizon 2020 projects that dive into the concept of orchestration, the SERRANO [18] project introduces an abstraction layer that reshapes edge, cloud and high performance computing resources into a unified infrastructure, as well as facilitating its automated and cognitive orchestration. Next, the CHARITY [19] project tackles the underlying issues of technological developments in the field of virtual reality, augmented reality and holography by utilizing new cloud computing architectures and the continuous and autonomous orchestration of computing and network resources. The RADON [20] project has the goal of delivering software faster, easier, and cheaper, by broadening the adoption of serverless computing technologies. Towards this end, it applies function-level scaling and billing, and automated orchestration and reuse of microservices and data pipelines. The ELEGANT [21] project tackles the underlying problems of utilizing IoT and Big Data technologies such as interoperability, reliability, safety, and security. It proposes to alleviate these hurdles with a framework that offers lightweight virtualization, automatic code extraction compatible with the utilized technologies, intelligent orchestration, and cybersecurity mechanisms, amongst others. In DICE [25], a Dicer deployment engine was developed and is the origin of the xOpera project. In this sense, the IEM component of PIACERE operates in the field of the deployment orchestration. On one hand, declarative technologies such as Terraform provide a definition of the final state of a deployment. On the other hand, imperative languages such as Ansible define a set of actions that need to be executed sequentially. The IEM is able to utilize these technologies that operate different stages of the application's deployment life cycle into a single component. These stages have been defined in previous steps of the PIACERE framework. Finally, the Gaia-X [9] project is a European initiated project that promotes the creation of the next generation of data infrastructure. An alignment of the IEM with this project would be beneficial for both in order to promote the reusability of both the code and the ideas behind.

2.2 Analysis of shortcomings

In the field of configuration management, there are two different approaches the tools propose for tackling this stage. The first one is to use an agent-based approach, in which each of the infrastructural devices holds an agent which communicates with an orchestrator and manages the configuration of the device. The second one is more decentralized, the infrastructural devices are left as they are, and the configuration is made through standard methods such as secure shell connections. The second approach is the one used by Ansible, which is the selected tool for the configuration within the PIACERE framework. It is less intrusive, straightforward and requires less configuration at the beginning. This is a big advantage, PIACERE focuses mainly on deployments performed from scratch.

The infrastructure provisioning is another important field of technologies for the IEM. In it, some of the tools focus exclusively on a single cloud provider. This becomes a big issue, given that the PIACERE framework aspires to interact with multiple providers, depending on the use case. For this reason, Terraform is more flexible since it can interact with multiple public and private cloud providers and is comprised of a myriad of plugins to perform different tasks in the field of the infrastructure provisioning. The redeployment of the infrastructure is also one of the IEM's main duties, the utilization of a technology such as Terraform facilitates the partial redeployment of certain parts of the infrastructure should it be needed. This might be the designated approach in bigger infrastructures in which downtime should be minimized. On the contrary, a full redeployment is also possible if the more phased approach is not required, and a starting with a blank slate is more appropriate.

Finally, the operationalization of a given deployment can be accomplished with container and deployment orchestration solutions. The first ones are focused on the deployment, monitoring, and orchestration of container-based applications. They have undergone a rapid adoption in industry, and in major cloud providers. On the other hand, the latter specializes in the operationalization of generic applications. However, these set of tools have not reached the same level of maturity of their container-based counterparts. The orchestration of the applications is obtained utilizing docker compose. However, this technology has been designed to be executed on the local environment, whereas the IEM requires the remote execution. The IEM takes care of this issue by targeting the remote docker socket available in the provisioned infrastructure.

One of the main threats in the implementation of this component is that the technologies that are utilized underneath are specifically tailored to be executed independently, hence they tend to follow the fail-fast approach which might not ideal for the integrated approach the IEM provides. Due to this, the IEM takes care of the execution lifecycle of these technologies and triggers an automatic redeployment when required, without having to bother the end user with these nuances. This is possible due to the fact the projects operationalized within PIACERE are idempotent. However, this kind of redeployments might be risky. For instance, in case a false positive is detected a series of iterative redeployments might be triggered, we minimize this risk by providing a threshold over which no more redeployments are executed. In addition, there might be a lingering root cause that can be trickier to identify due to the attempted redeployments performed by this tool, or a significant the infrastructure might incur into significant downtimes.

2.2.1 Applicability in the current project

The different tools and technologies analysed in this project are of paramount importance for the development of the IaC Execution Manager. Given that the IEM aspires to provide a common interface for other components of the PIACERE framework to execute their deployments, some of the tools analysed in this chapter have been selected to fulfil PIACERE's goals. The provisioning and configuration of infrastructural elements is one of the key values of the IEM, and this provisioning of these infrastructural elements is going to be done with Terraform. The selection of this tool has been done due to its extreme popularity, which carries a large community, and it is expected to be production ready. In addition, it can be used to orchestrate deployments in the most popular private and public cloud providers.

Next, the configuration of the afore mentioned infrastructural elements will be accomplished using Ansible. Similar to Terraform, Ansible provides a large community and a wide catalogue of plugins that can be used to accomplish all the needs that will arise within the PIACERE project. These two tools are considered to be extremely trustworthy, while also providing excellent

flexibility to reach the project's goals. Finally, the deployment of the use case's applications is to be achieved

2.3 PIACERE IaC Execution Manager goals

The PIACERE IaC Execution Environment is an important part of the PIACERE effort. The provided component – IEM (Infrastructure Execution Manager) – plays a central role in all workflows supported by PIACERE: it enables the IaC code to actually run. The primary goal of IEM is to provide a tooling around chosen, supported IaC execution tools, such as Ansible or Terraform. This tooling would understand and handle the differences between the different execution tools and would be able to extract information relevant to the PIACERE framework. This information includes the details on the IaC run process itself as well as its results (e.g., the created cloud resources (such as VMs), time taken to be created, among others) and all the relevant modifications made in the infrastructure layer.

A non-goal for IEM is to provide a new tool to run IaC – only known, existing, open-source tools will be used with IEM. IEM will not replace any of these tools, nor is its goal to directly enhance their capabilities.

2.4 PIACERE IaC Execution Manager's relevance to use cases

The IEM is fundamental for reaching PIACERE's goals, and so it is for the use cases. The IEM's overarching goal is to provide a common interface for the deployment of different Infrastructure as Code technologies in a unified way. In addition, it will offer metrics and metadata regarding these deployments to other components within the PIACERE framework. The use cases' needs in terms of technology have been elicited in deliverable D7.1 [9] and the IEM has been designed and implemented with those needs in mind. Hence, this component serves as main point for managing the different deployment scenarios that will be implemented within the PIACERE framework, and it incorporates all the required technologies to interact with the different public and private cloud providers required by the use cases.

3 Implementation

This section is devoted to the details regarding the implementation of the IaC Execution Manager. In Section 3.2 the requirements that have been identified by the project for the IEM to address are presented. Next, Section 3.3 showcases the functional description and how the IEM fits in the overall PIACERE architecture. Section 3.4 explains the technical description of the project, including the architecture of the prototype, a description of each of the components within the IEM, and the technical specifications of the project.

3.1 Changes in v2

In this second iteration, the IEM component comes with new features mainly focussed on the usability, and development of enhanced functionalities for the use cases and the rest of the PIACERE components. In addition, the integration with the overall PIACERE ecosystem has been the cornerstone that has driven the development efforts during the year.

One of the most remarkable features of this year has been the implementation of new Infrastructure as Code engines within the IEM core functionalities. Previously, the prototype version was developed around the integration with the AWS public cloud provider. However, much of the efforts of this iteration have focussed on the integration of the OpenStack private cloud provider. In addition, this orchestrator is the one selected by other PIACERE tools, hence this effort has contributed towards a better integration with the overall architecture. The outcome of this effort is that the PIACERE demo example is fully supported by the IEM in the OpenStack environment. Some of the features that have been successfully implemented are the provisioning of different flavours of virtual machines, the associated networking within and towards the created infrastructural devices, and the appropriate credential handling of the provisioned environment, amongst others.

Next, the very first prototype of the IEM relied on terraform itself for the configuration management of the various infrastructural devices with ansible. During this year, an intermediate structure has been adopted based on YAML configuration files for the integration and bundling of the different languages that the IEM is able to handle. This approach not only solves the current implementation of the different languages but also provides means for future technologies that may be necessary in the further iterations but are not yet adopted by the different tools of the PIACERE framework. On a similar note, this approach has been adopted by other tools such as the ICG to provide an integrated solution.

An important effort has been conducted for the implementation of the persistence layer. This feature facilitates the tracking of current and past deployments and their status. This way, the different problems, pitfalls, and successful executions that have taken place are logged and can be queried by the user for further research. This layer represents an important advancement for other tools such as the IDE, which utilizes for providing appropriate feedback to the stakeholders.

Credential handling has driven much attention during this second year not only for the IEM but also in the consortium. In particular, the different cloud providers, and infrastructural devices the IEM interacts with have their differences in the way they handle their credentials. Firstly, a generalization of the different credentials was required and has been adopted for current and future orchestrators. Secondly, the way these credentials are safely handled and disposed needed to be implemented. Finally, the integration with the PRC, another pivotal component of PIACERE, has been conducted.

Finally, many more features have been implemented during this year. Some of the most relevant ones are related with the testing, code quality and coverage of the developed software. The

documentation is now integrated with the same repository so that the IEM becomes more usable by the consortium and third parties. Monitoring features have been implemented so that the runtime of the provisioned infrastructural devices can be tracked. Summarizing, the focus of this year has been on adding and enhancing functionalities required by the consortium and in the integration efforts required for running as part of the PIACERE ecosystem.

This deliverable follows the approach of a rolling deliverable, meaning that the content from the first prototype that remains relevant has been kept, and the different sections have been updated, and amended when required. The introduction in Section 1 has been updated reflecting changes in the content of the deliverable, and the document structure. In Section 2, the large majority of the state of the art has been referenced back to the previous deliverable, the orchestration in H2020 remains mostly the same but new references have been added at the end, and the shortcomings sections has been amended with the problems the IEM has tackled during this second iteration. Section 3 contains a lengthy explanation of the efforts and accomplishments of this second prototype. In addition, the requirements sections have been updated to reflect this work, and the relevant diagrams have been replaced or appended if required. The project skeleton subsection reflects the work that has been carried out towards the integration of the different IaC technologies and PIACERE components. In Section 4, delivery and usage, the information gathered from the first prototype still applies, but several figures have been replaced to offer a more accurate description of the current state, and the licensing information has been filled out. Finally, the conclusions have been updated, and an appendix with the main features of the IEM is provided.

3.2 Requirements covered by this prototype

The following table shows all the requirements that need to be addressed by the IEM throughout the PIACERE project, extracted from WP2 deliverable D2.1 [10] However, not all of them need to be necessarily completed at this stage of the project, as they might include interaction with other components or might have not been classified as high priority to be included in the first year of the project.

Table 1 - Requirements that need to be addressed by the IEM and its current fulfilment status.

Req ID	Description	Status	Requirement Coverage at M24
WP5.1-REQ1	The IEM shall allow redeployment and reconfiguration, both full and partial, as allowed by the used IaC technology.	Satisfied	The IEM covers this requirement in its current development status. However, it relies on the IaC code being idempotent so this must remain the case.
WP5.1-REQ2	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.	Satisfied	The IEM will cover this requirement with a relation database in which the related metrics for past and present executions of the given deployments will be registered.
WP5.1-REQ3	IEM should be able to execute IaC generated by ICG for selected IaC languages (e.g., Ansible/Terraform)	Satisfied	The IEM covers in its current form the execution of two different IaC technologies: Terraform and Ansible.
WP5.1-REQ4	IEM shall register the status of past and present executions	Satisfied	The IEM will cover this requirement with a relation database in which the related

	and enable an appropriate way to query it.		metrics for past and present executions of the given deployments will be registered.
WP5.1-REQ5	IEM should be able to communicate with the relevant actors (orchestrators, infrastructural elements) in a secure way.	Partially Satisfied	The IEM communicates with the relevant actors by utilizing the secrets provided by the PRC. However, this effort will continue throughout the second year of the project in order to guarantee that these secrets are used correctly.
WP5.1-REQ6	IEM should be able to utilize the required credentials in a secure way.	Partially Satisfied	The IEM stores is able to utilize the credentials in a secure way. However, further efforts in this direction will be undergone during the second year of the project.
WP5.1-REQ7	IEM should be able to clean up the resources being allocated.	Satisfied	This component is able to clean up the resources being allocated by the different deployments.
WP5.1-REQ8	IEM shall work against the production environment and the canary environment.	Satisfied	The IEM is able to utilize the production environment, mainly private and public providers. The integration with the canary environment will be tested during this year of the project.

3.3 Functional description

The IaC Execution Manager is the component in charge of kicking off the different deployments taking place within the PIACERE framework. In addition, it oversees the subsequent redeployments and the finalization of the given deployments. In summary, the IEM is able to prepare and provision the infrastructure and install the corresponding software elements required by the deployment to run seamlessly.

One of its key features is its ability to execute different Infrastructure as Code technologies together, so the whole process of deployment and redeployment of the infrastructural elements is controlled in a centralized manner. For this prototype, the IEM can understand and execute two technologies covering different stages of the infrastructure deployment: Terraform for the provisioning of the different infrastructural elements required by the deployments in the different public and private cloud providers, and Ansible for the configuration of such infrastructural elements. In addition, the IEM has been designed so that additional IaC technologies that have not been considered as part of this prototype can be integrated in advance without cumbersome adjustments in its overall architecture.

Further releases of this component will include the ability to retrieve metrics about the status of past and present deployments. These metrics can be utilized by the different components that coexist in the PIACERE infrastructure to fulfil their goals.

3.3.1 Fitting into overall PIACERE Architecture

The IEM is the component of the PIACERE architecture that executes the DOML and the IaC code being created on previous stages of the PIACERE workflow. The component which interacts the most with the IEM is the Runtime Controller (PRC), as can be seen in Figure . The main interactions are as follows:

- The PRC communicates with the IEM to trigger a deployment. In order to do so, it hands over the following information:
 - The location of the deployment that is going to be executed by the IEM.
 - The commit id referencing the version of the deployment that needs to be executed.
 - The secrets that are necessary for the execution of the given deployment. These secrets will be treated and stored appropriately.
- The IEM communicates with the IaC Repository to get the deployment referenced by the PRC. This deployment is the one that will be triggered by the IEM.
- The IEM will be able to store metadata and metrics about past and present deployments for the rest of the elements in the architecture to query and utilize for their goals.

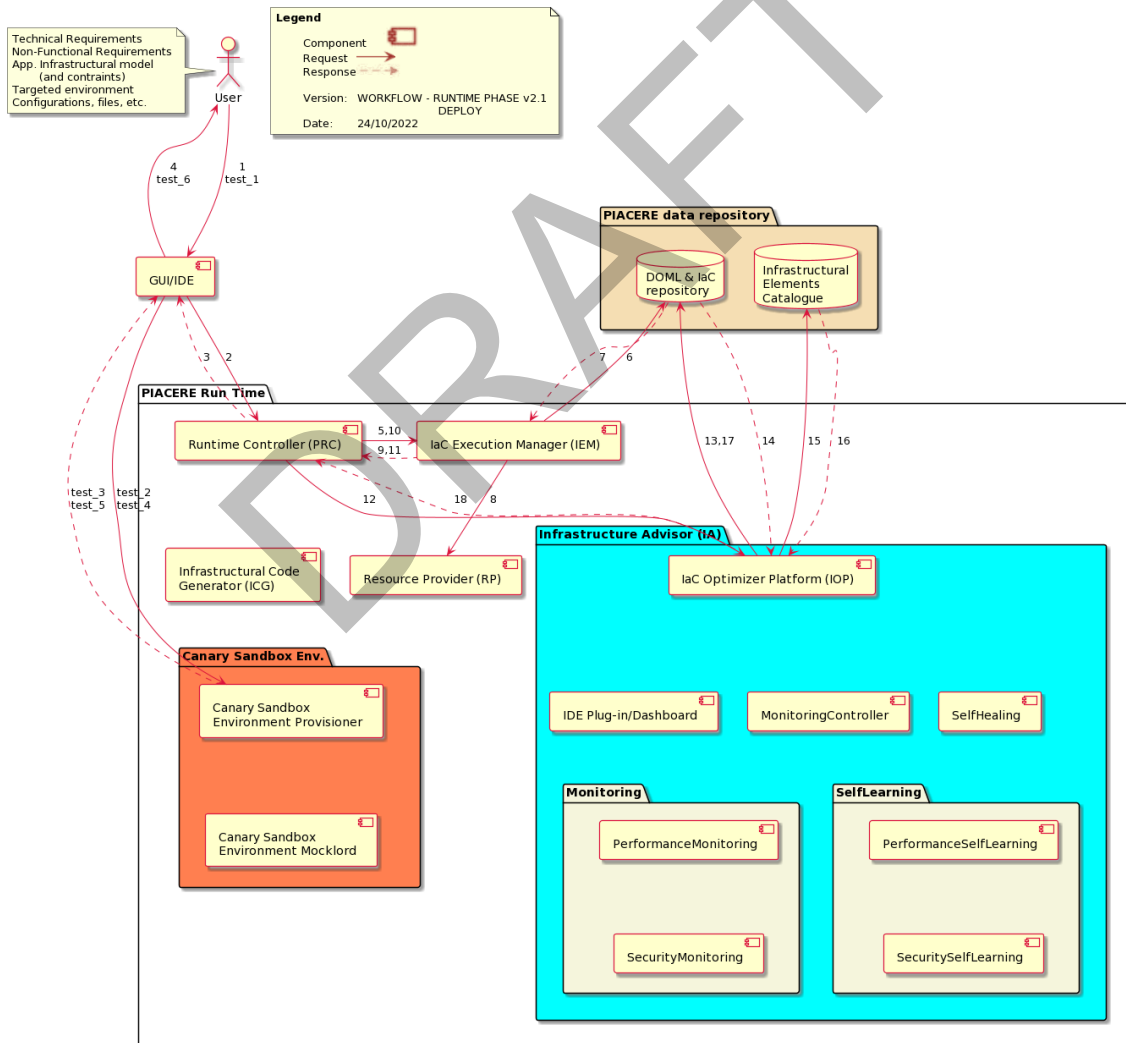


Figure 1 – PIACERE Runtime Deploy Diagram (version 2.1)

In the following diagram, the PIACERE runtime monitoring workflow diagram is displayed. The IEM is yet again pivotal in this endeavour. This is because it provides means to other components

in the architecture for triggering specific actions on already running components. This way, the monitoring and self-healing of the architecture can be accomplished.

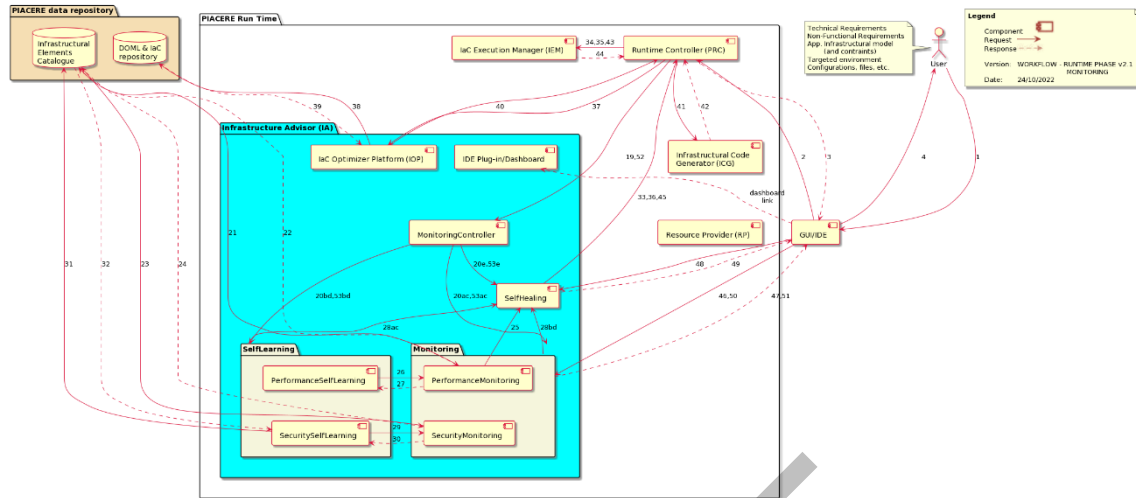


Figure 2 - PIACERE Runtime Monitoring (version 2.1)

3.4 Technical description

This subsection contains the details regarding the technical description of the IEM component. First, the overall architecture of the IEM component is showcased. Then, each component in the IEM is described in detail. Finally, the technical specifications of each of the components are described in detail.

3.4.1 Prototype architecture

The architecture components that comprise the entirety of the IEM component are depicted in the following Figure . These is a REST API that is in charge of managing the interaction with the IEM. The Core of the system where the business logic resided and is able to forward the different actions appropriately. The Persistence component which is in charge of storing the metrics and metadata related with past and present deployments. Finally, the executors are able to understand the IaC code being forwarded to the IEM and execute it against the different public and private cloud providers. In the next subsection each of these components are explained in further detail.

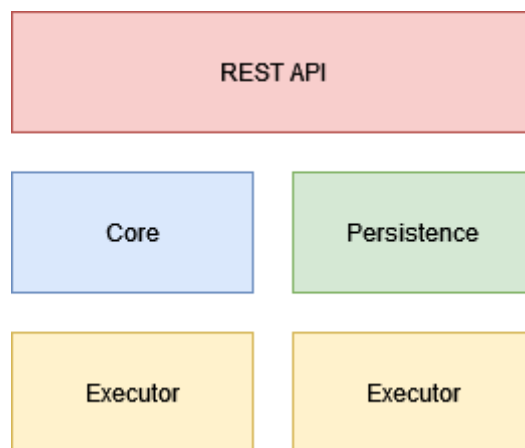


Figure 3 - Architecture of the different elements comprising the IEM Execution Manager

3.4.2 Component description

The first prototype of the IEM component is comprised of the subcomponents described above. In this section, we take a closer look at each and explain their functionality in further detail.

- **REST API:** this subcomponent is the entry point of all the requests that the IEM need to process. The interaction with this component is through this technology, and a OpenAPI specification file is provided so the interaction with it becomes as seamless as possible.
- **Core:** this subcomponent contains the business logic of the IEM component. It oversees forwarding the different calls of other PIACERE components appropriately. In addition, it is able to interact with other PIACERE components such as the IaC Repository, which is the component where the different deployments are stored for the IEM to utilize.
- **Persistence:** this subcomponent contains the persistence logic that the IEM component is going to utilize. It is a relational database that will provide the data required for the requests for information by other components. This is information and metadata for past and present executions of the different components. Most of the development for this subcomponent has been undergone during the second year of the project.
- **Executors:** the executors are the subcomponents in charge of the execution of the different technologies that the IEM supports. For this very first prototype, two different IaC technologies are supported: i) Ansible for the configuration of the different dependencies that the deployment requires, ii) Terraform for the provisioning of the infrastructural elements required for the deployments to be successfully executed, iii) docker for the application lifecycle management.

3.4.3 Component interactions

In this section the interaction of the IEM for two important PIACERE workflows are depicted and explained in detail. These are the start of a deployment and the request of the status of a given deployment. The former has been implemented for this very first prototype of the IEM component, with the exception of the persistence layer. The latter has been undergone during the coming year of the project and is depicted in this deliverable for a better understanding of the component duties.

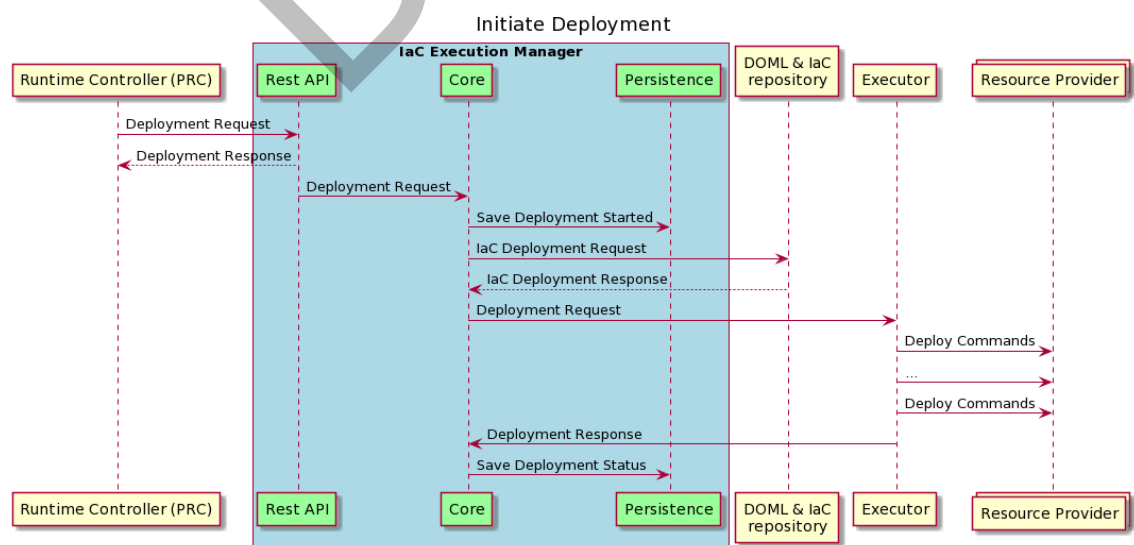


Figure 4 - Sequence Diagram for the process of kicking off a deployment

The Figure 4 above oversees the process of kicking off a deployment within the PIACERE ecosystem. A deployment is started by the Runtime Controller. Given that a deployment might take a long time to finalize, this has been identified as an asynchronous task and an immediate response is sent back to the PRC. For all it knows, the PRC will only have the information about the deployment being properly initialized, but it will not know how that given deployment has gone (this process is depicted and explained in further detail in the following Figure). The security of this communication has been developed during the coming year of the project. The request is forwarded to the Core subsystem, which will immediately be recorded in the persistence subcomponent (to be implemented in year two). Next, the Core subcomponent retrieves the appropriate deployment for the DOML & IaC Repository, only to execute it against the resource provider. This deployment is expected to be in a mixture of the two technologies currently supported by the PIACERE framework (Ansible and Terraform). Once the deployment is finalized, which might take a long time, the result of this deployment is persisted so it can be queried by other components.

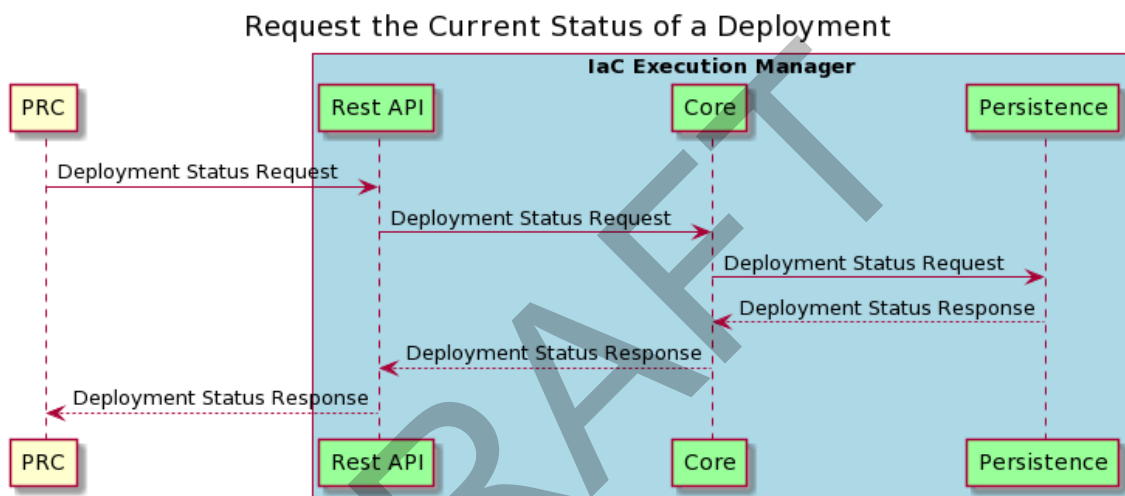


Figure 5 – Sequence diagram of the process of requesting the status of a given deployment

The Figure 5 above exemplifies the process of querying the status of a given deployment. This functionality has already been implemented as part of this prototype during the second year of the project. In this diagram, the PRC initiates the process of requesting the status of a past or present deployment. This petition is received by the REST API and forwarded to the core subcomponent. Then, the core subcomponent will retrieve this information from the persistence layer and return it to the Rest API, so the PRC can get an immediate response to its petition. This process could be complemented with a notification mechanism if required by the stakeholders. However, this has not been implemented as yet and might be something to consider in future releases if required by the users.

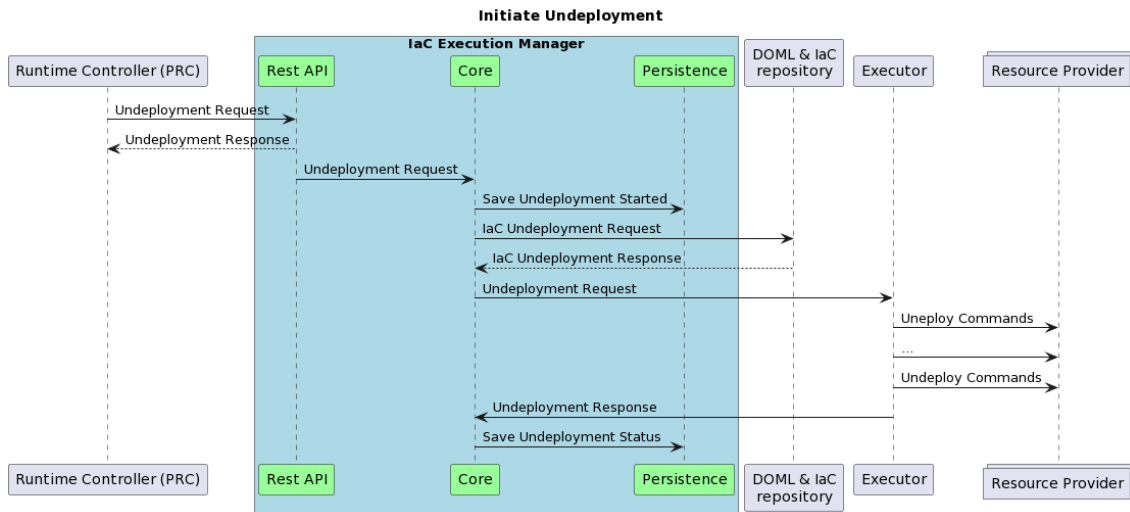


Figure 6 - Sequence Diagram for the process of tearing down a deployment

In the above diagram, the sequence of the tearing down a deployment in the given infrastructure is showcased. This process heavily relies on the capabilities of the infrastructure provider for destroying the provisioned servers. The workflow is analogous to the deployment.

3.4.4 Technical specifications

The prototype has been developed in the Python programming language, specifically version 3.9.5. It has been selected because python is very proficient at interacting with the currently used IaC technologies (Terraform, Ansible), and provides an easy manner to add additional technologies in the future.

The input and output interactions of this component are supported by FastAPI, which provides a myriad of functionalities for the implementation of REST interfaces, which is the primary way of interacting with the IEM component. In addition, it provides functionalities for providing an OpenAPI implementation that can be used by other components in the PIACERE infrastructure, as it not only provides an easy way to understand the inputs and outputs required by this component, but also an automatic way to generate the server and client sides if desired.

The persistence layer is in essence a relation database. This piece of the component has been completed during the second year of the project. The persistence layer has been implemented using SQLite, and it contains a set of relational tables that will store the metrics, metadata, and information related to past and presents deployments.

The IaC code that have been selected to use within the PIACERE framework have been Terraform and Ansible. The former is a well-known technology utilized in the field of infrastructural device provisioning and can interact with a large variety of public and private cloud providers (e.g. AWS, Azure, Google Cloud). The latter, on the other hand, is an established tool in industry that is commonly used for the configuration of the infrastructural devices required for the deployment. It has a myriad of modules that can be used for the different nuances that comprise a software project deployment such as dependency management, services configuration, and configuration management.

Access to the IEM is hardened with an API key that needs to be fed into the system every single time that an action is required to be taken.

Finally, the actual delivery of the component in a containerized manner, with the docker technology.

3.4.5 Project skeleton

One of the benefits of using the IEM for the project's life-cycle management of projects is that it provides an abstraction over the provisioning, configuration, and application orchestration in a common tool. To this end, the various underlying technologies utilized to accomplish this endeavor are glued together in a common bundle, and the IEM oversees the seamless sequential execution of them. In its current state, the IEM has two different processing engines: the 1) ansible configuration management, and the 2) terraform provisioning tools. The interaction and swapping between these two processing engines are handled through simple configuration files written in YAML.

Name	Last commit	Last update
📁 nginx	Reset to ICG output	2 months ago
📁 piacere_monitoring	removes defaults and force to have value	1 month ago
📁 piacere_monitoring_requirements	fixes the env variable or default	1 month ago
📁 terraform	simplify security group	1 month ago
📄 config.yaml	splits requirements from monitoring	1 month ago

Figure 7 - Sample project with various sequential stages.

In Figure 7 the root folder of a project is displayed; the different folders represent different stages that are going to be sequentially executed. The information on the order is specified in the configuration file that can be seen in Figure 8.

```

1  ---
2  iac:
3  - terraform
4  - piacere_monitoring_requirements
5  - piacere_monitoring
6  - nginx
7  ...

```

Figure 8 - Configuration file stating the sequential execution order of the deployment.

This file is very easy to be consumed both humans and programmatically. If we dive into the nginx stage a similar structure can be seen, Figure 9 showcases that this stage contains the configuration information for an ansible executor, and the credentials that are to be used during this execution in the form of a jinja file.





 config.yaml	Reset to ICG output
 inventory.j2	Reset to ICG output
 main.yml	Move nginx.yaml to main.yml
 ssh_key.j2	Reset to the new ICG content

Figure 9 - nginx stage folder structure.

Finally, a configuration file can be found in this folder that will specify the IEM how to handle this stage. This information can be seen in Figure 10, and the following information is detailed:

- Input: The various inputs that are to be fed into this stage in the form of environment variables.
- Output: The various outputs that are to be used by subsequent stages, that will be written into environment variables.
- Engine: the specific engine that is to be used within this stage, in this particular example the ansible execution engine will be used.

```

2  ---
3  input:
4    - instance_ip_vm1
5    - instance_server_private_key_user1
6  output: []
7  engine: ansible
8  ...

```

Figure 10 - Configuration information for the nginx stage.

Using this methodology and syntax the IEM is able to handle projects written in various provisioning, configuration, and orchestration languages that are currently in use alongside future engines that are not yet envisioned or even available at the time of writing this document.

3.4.6 Public and private cloud provisioners

The IEM has been designed so its execution is agnostic from the cloud provider, the credentials are fed into the IEM in the deployment request as can be seen in Figure 11. In its present form, credential handles for Azure, and AWS public cloud providers, and the OpenStack public cloud providers are in place.

```
"credentials": {  
  "aws": {  
    "access_key_id": "string",  
    "secret_access_key": "string"  
  },  
  "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"  
  }  
}
```

Figure 11 - credentials section that are fed into the IEM to kick off a deployment.

During this second year of the project, the IEM has been successfully tested against both OpenStack, and AWS. In addition, the validation has been successfully performed against the OpenStack public cloud provider.

3.4.7 Challenges and pitfalls

The integration of different provisioning, configuration, and application orchestration tools comes at a cost. Each of different technologies are specifically tailored to be executed independently and trying to glue them together into a common interface requires tackling challenges that are not present otherwise.

A good example of this behavior is the ansible engine integrated within the IEM. Ansible tends to fail easily should there be connectivity issues, or any other unexpected problem during its execution. This is not a problem when running it interactively, as an idempotent definition of the ansible project only would require another run to finish successfully in most cases. However, this behavior can be particularly inconvenient when used as part of a framework, as a failure in any stage would result in a failure of the whole deployment, which can take a long time to reproduce. There are many possible reasons for a failure in the deployment, and the IEM can only do its best with some of them, such as connectivity issues. Furthermore, other types of failures such as the lack of available resources should be evaluated independently. Due to this, the IEM is able to detect these problems during runtime and keep retrying for a predefined number of times to overcome them without requiring manual intervention, hence the user is unaware of such failures resulting in a better user experience. This kind of issues will be reported in the database so the user can always track the potential failures.

Being able to handle the credentials in a unified secure way can be cumbersome in application having to be used by multiple users. In this regard, the IEM does not persist any credentials into physical storage, instead treats them as environment variables that become isolated to the different users that require the unified deployment capabilities this tool offers. This approach

has been taken because the persistence of credentials such as certificates might raise some challenges in terms of resource isolation that environment variables do not have. In addition, the orchestrators the IEM is communicating with such as AWS, Azure, or OpenStack promote the use of environment variables during runtime. Finally, the risk of certificates being submitted to source code repositories, even though it is something the user should never do, and it would be entirely his fault, is yet another thing not to be concerned by using environment variables.

DRAFT

4 Delivery and usage

This section offers information on the package itself. First, information about the structure of the repository is provided. Then, instructions on how to install the package are offered. Thirdly, a manual on how to utilize the IEM component is explained in detail. Finally, licensing information and downloading instructions are provided.

4.1 Package information

This section gives an overview on the structure of the IEM component. The root structure for the component is showcased in the following image, information about the most relevant files and folders are then explained.

Name	Last commit	Last update
doc/sequence-diagrams	Update 51-request-deployment-status.puml	3 hours ago
iem	deploy terraform script	1 week ago
.gitignore	update .gitignore	1 month ago
.gitlab-ci.yml	fix ci	5 months ago
Dockerfile	enable 0.0.0.0	3 months ago
README.md	docker build stage for iem	5 months ago
openapi.json	deploy terraform script	1 week ago
sonar-project.properties	fix ci	5 months ago

Figure 12 – Root folder for PIACERE's IEM component

The “doc” folder contains the documentation regarding the IEM component implementation. In particular the sequence diagram that explain the interaction of the IEM with the rest of the components resided under this folder.

Build	Test	Scan
package	test	sonarqube-c...

Figure 13 – Continuous Integration and Deployment pipeline for the IEM project

Next, the “.gitlab-ci.yml” file provides continuous integration and deployment details about the project. In this file three main stages are defined.

- The “package” stage is in charge of assuring that the IEM is able to be containerized following the instructions in the “Dockerfile” file, which resides under this very same root folder.
- Next, the “test” stage executes all the test files that have been defined in the project, should a test fail, the entire pipeline would stop and the developer should address the issue immediately.

- Finally, the “scan” stage will utilize the sonarqube tool to scan the source code for issues and provide an integrated dashboard with all the information regarding the project. This dashboard is showcased in the following image (Figure).

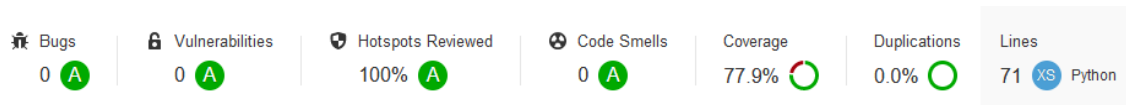


Figure 14 – SonarQube dashboard of the IEM component

As it can be seen the project is healthy and the coverage is up to 77.9% which is appropriate for this stage of the development phase. However, further efforts will be done in order to raise the coverage threshold further.

Next, the “openapi.json” file provides information about how to interact with the IEM, this file will be utilized by other components in order to assure the communication is as seamless as possible. Further information about this file is explained in Section 4.3.

Finally, the “iem” folder contains the whole source code for the IEM component alongside the tests that make sure the code works correctly. The central code file is the “main.py” file located under this folder and is the one in charge of importing and calling the necessary modules for running the IEM.

4.2 Installation instructions

The IEM prototype can be found in Tecnalía’s GitLab repository (download instructions at the end), and it is also provided as a compressed folder. There are a few files that are of paramount importance for getting the IEM prototype up and running. The first one is the “requirements.txt” file, which offers an up-to-date list of the IEM dependencies alongside their specific version. It would be better to install these requirements in a virtual environment in order not to mess with the local installation of similar packages. There are many different ways to start a virtual environment, and specific instructions are out of the scope of this deliverable. Hence, in this document brief instructions towards the installation and use of a tool for creating virtual environments (virtualenv) are provided. The installation of the virtualenv can be accomplished with the following command.

```
vagrant@piacere:~$ sudo pip3 install --upgrade pip
Collecting pip
  Downloading pip-21.3.1-py3-none-any.whl (1.7 MB)
    | 1.7 MB 2.9 MB/s
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 20.0.2
    Not uninstalling pip at /usr/lib/python3/dist-packages, outside environment /usr
    Can't uninstall 'pip'. No files were found to uninstall.
Successfully installed pip-21.3.1
vagrant@piacere:~$ sudo pip3 install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.10.0-py2.py3-none-any.whl (5.6 MB)
    | 5.6 MB 2.9 MB/s
Collecting backports.entry-points-selectable>=1.0.4
  Downloading backports.entry_points_selectable-1.1.0-py2.py3-none-any.whl (6.2 kB)
Collecting filelock<4,>=3.2
  Downloading filelock-3.3.2-py3-none-any.whl (9.7 kB)
Collecting platformdirs<3,>=2
  Downloading platformdirs-2.4.0-py3-none-any.whl (14 kB)
Requirement already satisfied: six<2,>=1.9.0 in /usr/lib/python3/dist-packages (from virtualenv) (1.14.0)
Collecting distlib<1,>=0.3.1
  Downloading distlib-0.3.3-py2.py3-none-any.whl (496 kB)
    | 496 kB 19.4 MB/s
Installing collected packages: platformdirs, filelock, distlib, backports.entry-points-selectable, virtualenv
Successfully installed backports.entry-points-selectable-1.1.0 distlib-0.3.3 filelock-3.3.2 platformdirs-2.4.0 virtualenv-20.10.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

Figure 15 - Excerpt showing the installation steps for virtualenv in a linux box

Now, with the virtualenv executable, a virtualenv can be created by running the following command.

```
vagrant@piacere:~$ virtualenv env --python=python3
created virtual environment CPython3.8.10.final.0-64 in 303ms
  creator CPython3Posix(dest=/home/vagrant/env, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/vagrant/.local/share/virtualenv)
  added seed packages: pip==21.3.1, setuptools==58.3.0, wheel==0.37.0
  activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
```

Figure 16 - Excerpt showing the creation of a virtual environment

Then, in order to start this freshly created environment, the following snippet shows how to do this and start working with the new environment.

```
vagrant@piacere:~$ cd env/
vagrant@piacere:~/env$ source bin/activate
(env) vagrant@piacere:~/env$
```

Figure 17 - Excerpt showing the activation of the freshly created virtual environment

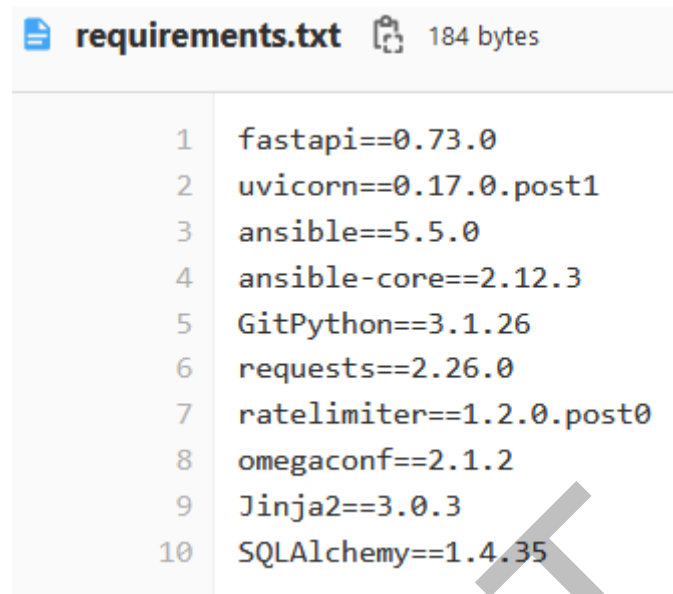
It becomes useful to know to get out of the environment which is called deactivate. Next, an example on how to accomplish this is shown.

```
(env) vagrant@piacere:~/env$ deactivate
vagrant@piacere:~/env$
```

Figure 18 - Excerpt showing how to finalize or deactivate a virtual environment

Now that a proper virtual environment has been created and it is ready to be used, the “requirements.txt” becomes handy. In the following image it can be seen the content of it, which the precise libraries alongside their version that need to be installed in order to run the IEM

prototype. It also showcases how to install these requirements with the python package manager.

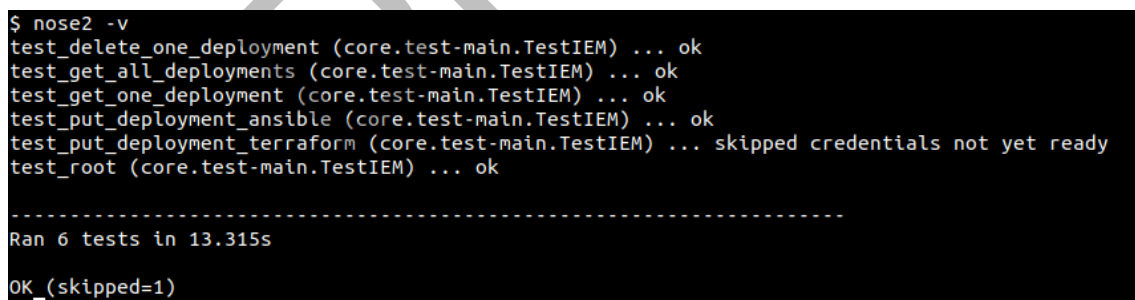
A screenshot of a file editor window showing a file named 'requirements.txt' with a size of 184 bytes. The file contains ten lines of Python package requirements, each on a new line, numbered 1 through 10. The requirements are: fastapi==0.73.0, uvicorn==0.17.0.post1, ansible==5.5.0, ansible-core==2.12.3, GitPython==3.1.26, requests==2.26.0, ratelimiter==1.2.0.post0, omegaconf==2.1.2, Jinja2==3.0.3, and SQLAlchemy==1.4.35.

```
requirements.txt 184 bytes
1 fastapi==0.73.0
2 uvicorn==0.17.0.post1
3 ansible==5.5.0
4 ansible-core==2.12.3
5 GitPython==3.1.26
6 requests==2.26.0
7 ratelimiter==1.2.0.post0
8 omegaconf==2.1.2
9 Jinja2==3.0.3
10 SQLAlchemy==1.4.35
```

Figure 19 - Excerpt showing the content of the requirements.txt file and how to install them

Unfortunately, not all the requirements can be centralized in this manner since the Terraform client needs to be installed separately. Please refer to the official documentation¹ to get this done. At the time of writing this document, the IEM component makes use of the local AWS credentials. Similarly, please refer to the official documentation to let the IEM use these local credentials.

At this stage, all the IEM dependencies should be installed and ready to be used. In order to make sure that this is in fact the case, the following image shows how to run the different tests that make sure the prototype is properly working.

A screenshot of a terminal window showing the output of a 'nose2 -v' command. The output lists six test cases: test_delete_one_deployment, test_get_all_deployments, test_get_one_deployment, test_put_deployment_ansible, test_put_deployment_terraform, and test_root. The first five tests passed with 'ok', while the sixth test was skipped due to 'credentials not yet ready'. The summary shows 'Ran 6 tests in 13.315s' and 'OK (skipped=1)'.

```
$ nose2 -v
test_delete_one_deployment (core.test-main.TestIEM) ... ok
test_get_all_deployments (core.test-main.TestIEM) ... ok
test_get_one_deployment (core.test-main.TestIEM) ... ok
test_put_deployment_ansible (core.test-main.TestIEM) ... ok
test_put_deployment_terraform (core.test-main.TestIEM) ... skipped credentials not yet ready
test_root (core.test-main.TestIEM) ... ok

-----
Ran 6 tests in 13.315s

OK (skipped=1)
```

Figure 20 - Excerpt showing a successful execution of the test of the IEM

After checking that all the tests have run successfully, the IEM is ready to be executed with the uvicorn server. The following excerpt shows how to perform this. If everything goes right, this should open a web server in port 8000 of the local computer.

¹ <https://learn.hashicorp.com/tutorials/Terraform/install-cli>

```

$ uvicorn main:app --reload
INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:      Started reloader process [21081] using watchgod
INFO:      Started server process [21083]
2021-11-03 11:07:00,834 INFO      Started server process [21083]
INFO:      Waiting for application startup.
2021-11-03 11:07:00,835 INFO      Waiting for application startup.
INFO:      Application startup complete.
2021-11-03 11:07:00,835 INFO      Application startup complete.

```

Figure 21 - Excerpt showing how to kick off the IEM

At this stage, the IEM development environment should be up and running. However, the IEM component will be deployed containerized with the docker framework. There is a Dockerfile that helps in making this happen. The following excerpt shows the content of this file. This Dockerfile is based on the official image provided by the terraform team. Given the installation of this particular tool can be the most time consuming and error prone, we resort to this image in order to achieve better quality and efficiency.

```

Dockerfile 1.02 KIB
1 FROM hashicorp/terraform:1.1.4
2
3 ARG API_KEY
4
5 ENV API_KEY=$API_KEY
6 ENV IEM_HOME=/opt/iem/
7
8 COPY src/resources/ansible.cfg /etc/ansible/ansible.cfg
9 COPY requirements.txt /tmp/requirements.txt
10 COPY src ${IEM_HOME}src
11 COPY main.py ${IEM_HOME}main.py
12
13 RUN apk add py3-pip cargo g++ python3-dev file libffi-dev openssl-dev bash python3=3.9.13-r1 gnupg
14 RUN pip3 install -r /tmp/requirements.txt
15
16 # RUN adduser -h ${IEM_HOME} -S -D iem
17 COPY certs/config ${IEM_HOME}.ssh/config
18 COPY certs/id_rsa ${IEM_HOME}.ssh/id_rsa
19 COPY certs/id_rsa.pub ${IEM_HOME}.ssh/id_rsa.pub
20 RUN adduser -h ${IEM_HOME} -S -D iem && \
21     chown -R iem ${IEM_HOME} && \
22     chmod 0700 ${IEM_HOME}.ssh && \
23     chmod 0644 ${IEM_HOME}.ssh/config && \
24     chmod 0600 ${IEM_HOME}.ssh/id_rsa && \
25     chmod 0644 ${IEM_HOME}.ssh/id_rsa.pub
26 USER iem
27 RUN ansible-galaxy collection install community.general
28 COPY roles.yml /tmp/roles.yml
29 RUN ansible-galaxy install -r /tmp/roles.yml
30
31 ENTRYPOINT ["/usr/bin/env"]
32 WORKDIR ${IEM_HOME}
33 CMD /usr/bin/uvicorn main:app --host 0.0.0.0
34 EXPOSE 8000

```

Figure 22 - Excerpt showing the Dockerfile that will be used to generate the containerized image of the IEM

From this point, the docker image can be generated with the build command, and it gets tagged with the IEM name. Next, an instance of that image is run exposing port 8000 on the local computer.

Table 2 - Excerpt showing the built and execution of the IEM container

```

$ docker build --build-arg API_KEY=$API_KEY -t optima-piacere-docker-dev.artifact.tecnalia.com/wp5/iem-api:yl .

```

```
$ docker run -p 8000:8000 optima-piacere-docker-
dev.artifact.tecnalia.com/wp5/iem-api:y1
```

4.3 User Manual

This subsection gives an overview on how the communication with the IEM should take place. In particular, this is detailed in an OpenAPI specification file which different components can adhere to, in order to utilize the different functionalities provided by the IEM.

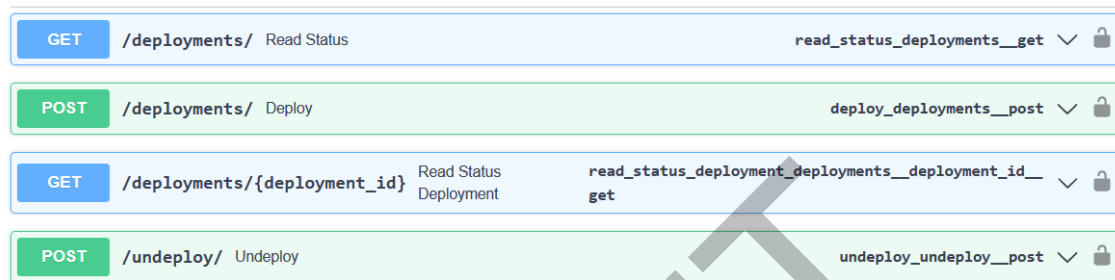


Figure 23 - OpenAPI specification for the interaction with the IEM component

The Figure above showcases a screenshot of the OpenAPI specification file that resides in the IEM's GitLab repository. For the first prototype, four endpoints have been defined for the interaction of PIACERE components with the IEM. The details on how these endpoints provide to the components is detailed below:

- **GET /deployments/**: it provides information about all the deployments that are currently taking place within the PIACERE framework.
- **POST /deployments/**: kick off a deployment, if the deployment has already been started in a previous iteration, it updates the given deployment with the new configuration. The figure below showcases the request body to be used within this deployment.

```
{
  "deployment_id": "string",
  "repository": "string",
  "commit": "string",
  "aws_access_key_id": "string",
  "aws_secret_access_key": "string",
  "private_key": "string"
}
```

Figure 24 - Request Body for the PUT /deployments/ endpoint

- **GET /deployments/{deployment_id}**: it yields detailed information about the status of a given deployment. The deployment to be retrieved should be passed as a path parameter.
- **POST /deployments/{deployment_id}**: the deployment specified in the `deployment_id` path parameter will be torn down.

In the second prototype of the IEM tool all these endpoints have become fully functional and tested throughout the second year of the project. They also provide other components with connection information with the IEM from early on the project.

In terms of the source code, the central point for the IEM is the “main.py” file. There, all the endpoints necessary to interact with this component are defined. The following excerpt showcases how this is obtained using the FastAPI framework.

```
@app.get("/deployments/{deployment_id}", response_model=DeploymentResponse)
async def read_status_deployment(deployment_id: str):
    d = DeploymentResponse(
        deployment_id='3',
        status='ok',
        startDate='2021-09-28T08:16:50.270Z',
        updateDate='2021-09-28T08:16:50.270Z',
        metadata={})
    return d

@app.put("/deployments/", status_code=status.HTTP_201_CREATED, response_model=BaseResponse)
async def deploy(d: DeploymentRequest, background_tasks: BackgroundTasks):
    background_tasks.add_task(iem.deploy, d.deployment_id, d.repository, d.commit)
    return BaseResponse(message="Deployment Request Created")
```

Figure 25 - Excerpt showing the main.py file in which the different endpoints for the IEM reside

This very same file also contains the definition for the inputs and outputs that will be necessary to provide the different endpoints. This way the OpenAPI is able to provide accurate information for other components to use the IEM interfaces. This information can be found in further detail in the “utils.py” file, in which the different responses are specified, as can be seen in the following excerpt.

```
class BaseResponse(BaseModel):
    message: str

class DeploymentResponse(BaseModel):
    deployment_id: str
    status: str
    startDate: datetime
    updateDate: datetime
    metadata: Dict

class DeploymentRequest(BaseModel):
    deployment_id: str
    repository: str
    commit: str
    aws_access_key_id: Optional[str] = None
    aws_secret_access_key: Optional[str] = None
    private_key: Optional[str] = None
```

Figure 26 - Excerpt showing the class holding the models used for the communication with the REST API

Finally, the prototype can be invoked using uvicorn, which is a ASGI (Asynchronous Server Gateway Interface) server implementation, as can be seen in the following excerpt.

```
$ uvicorn main:app --reload
INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:      Started reloader process [21081] using watchgod
INFO:      Started server process [21083]
2021-11-03 11:07:00,834 INFO      Started server process [21083]
INFO:      Waiting for application startup.
2021-11-03 11:07:00,835 INFO      Waiting for application startup.
INFO:      Application startup complete.
2021-11-03 11:07:00,835 INFO      Application startup complete.
```

Figure 27 - Excerpt showing how to initialize the IEM with the uvicorn server

This command will open a web server, the OpenAPI specification, and the documentation in the following addresses respectively.

Table 3 - Excerpt showing the relevant url's of the IEM

http://127.0.0.1:8000
http://127.0.0.1:8000/docs
http://127.0.0.1:8000/redoc

4.4 Licensing information

This component is offered under Apache 2.0 license. Detailed information can be found in the GitLab repository.

<https://git.code.tecnalia.com/piacere/public/the-platform/iem/-/blob/y1/LICENSE>

4.5 Download

The source code for the IEM prototype is available Tecnalia's GitLab repository. In order to get all the necessary files to utilize it, use the following link:

<https://git.code.tecnalia.com/piacere/private/t51-iem/iem>

5 Conclusions

This document revolves around the second prototype of the IaC Execution Manager (IEM). The goal of this platform is to plan, prepare, and provision the infrastructural and software elements for an application to seamlessly run.

In this second prototype the IEM is able to utilize heterogeneous Infrastructure as Code technologies, as Ansible and Terraform, comprised in a common interface, to successfully operationalize the different use cases in the PIACERE framework. The IEM is a key piece of the PIACERE framework, since it is the component in charge of interacting with different private and public cloud providers, with the orchestrator and with heterogeneous infrastructural elements. This way, the Infrastructure as Code being generated in previous stages of the PIACERE workflow is executed within this component.

The innovation of the IEM lies on unifying the provisioning, configuration, deployment, and orchestration stages of the application life cycle in a single component, and with a common interface. Hence, PIACERE is able to singlehandedly manage the application life cycle utilizing multiple technologies for the operationalization of applications in a common way.

As part of this second prototype, the different endpoints that are used by the IEM have been secured with token-based technologies. In addition, a persistence layer has been implemented to provide other components in the PIACERE framework with information and data about past and present deployments. At the moment, this prototype has focused on two well-known technologies (i.e., Terraform and Ansible) that have been identified as key by the use cases, but it is designed so that other relevant technologies can be easily integrated in future releases (e.g., docker-compose, Kubernetes) if desired.

Future iterations of this prototype will include a docker engine for managing the lifecycle of the applications to be deployed by the use cases. In addition, the possibility of triggering specific actions for the self-healing of the infrastructure will be supported. Finally, it will focus on the reliability of the whole component and its different endpoints, and on the integration with the other different PIACERE components within the infrastructure. In addition, the use of additional public and private cloud providers may be necessary.

6 References

- [1] PIACERE Consortium, “D5.1 IaC execution platform prototype,” 2021.
- [2] HashiCorp, «Packer,» [En línea]. Available: <https://www.packer.io/>. [Último acceso: 15 11 2022].
- [3] SERRANO, “TRANSPARENT APPLICATION DEPLOYMENT IN A SECURE, ACCELERATED AND COGNITIVE CLOUD CONTINUUM,” [Online]. Available: <https://cordis.europa.eu/project/id/101017168>. [Accessed 9 11 2021].
- [4] CHARITY, “Cloud for Holography and Cross Reality,” [Online]. Available: <https://cordis.europa.eu/project/id/101016509>. [Accessed 9 11 2021].
- [5] RADON, “Rational decomposition and orchestration for serverless computing,” [Online]. Available: <https://cordis.europa.eu/project/id/825040>. [Accessed 9 11 2021].
- [6] ELEGANT, “Secure and Seamless Edge-to-Cloud Analytics,” [Online]. Available: <https://cordis.europa.eu/project/id/957286>. [Accessed 9 11 2021].
- [7] DICE, “Data Infrastructure Capacity for EOSC,” [Online]. Available: <https://cordis.europa.eu/project/id/101017207>. [Accessed 26 11 2021].
- [8] Gaia-X, [En línea]. Available: <https://www.data-infrastructure.eu/GAIAX/Navigation/EN/Home/home.html>. [Último acceso: 15 11 2022].
- [9] PIACERE Consortium, “D7.1 PIACERE use case definition,” 2021.
- [10] PIACERE Consortium, «D2.1 PIACERE DevSecOps Framework Requirements specification, architecture and integration strategy,» 2021.
- [11] Apache Brooklyn, [Online]. Available: <https://brooklyn.apache.org/>. [Accessed 8 11 2021].
- [12] XLAB, [Online]. Available: <https://github.com/xlab-si/xopera-opera>. [Accessed 26 11 2021].
- [13] HashiCorp, [Online]. Available: <https://www.terraform.io/>. [Accessed 8 11 2021].
- [14] Apache Hadoop, “Apache Hadoop YARN,” [Online]. Available: <https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html>. [Accessed 8 11 2021].
- [15] AWS, “AWS CloudFormation,” [Online]. Available: <https://aws.amazon.com/es/cloudformation/>. [Accessed 8 11 2021].
- [16] CFEngine, “CFEngine,” [Online]. Available: <https://cfengine.com/>.
- [17] CHEF, “CHEF,” [Online]. Available: <https://www.chef.io/products/chef-infra>.

- [18] Mesosphere Inc., “Marathon - A container orchestration platform form Mesos and DC/OS,” [Online]. Available: <https://mesosphere.github.io/marathon/>. [Accessed 8 11 2021].
- [19] Cloudify Platform Ltd., “Multi Cloud Orchestration: Turn Glue Code Into Certified Environments,” [Online]. Available: <https://cloudify.co/>. [Accessed 8 11 2021].
- [20] OpenStack, “OpenStack,” [Online]. Available: <https://www.openstack.org/>. [Accessed 8 11 2021].
- [21] The Linux Foundation, “Production-Grade Container Orchestration,” [Online]. Available: <https://kubernetes.io/>. [Accessed 8 11 2021].
- [22] puppet, “puppet,” [Online]. Available: <https://puppet.com/>.
- [23] Rancher Labs, “Rancher,” [Online]. Available: <https://rancher.com/>. [Accessed 26 11 2021].
- [24] Red Hat, “Red Hat Ansible,” [Online]. Available: <https://www.ansible.com/>.
- [25] SaltStack Enterprise, “Salt Project,” [Online]. Available: <https://saltproject.io/>.
- [26] Spinnaker, “Spinnaker,” [Online]. Available: <https://spinnaker.io/>. [Accessed 8 11 2021].
- [27] URBANAGE Consortium, “URBANAGE_D3.1_Data Management Layer,” 2022.
- [28] Terraform, “Terraform,” 3 3 2022. [Online]. Available: Terraform.
- [29] Docker Inc, “Swarm mode overview,” [Online]. Available: <https://docs.docker.com/engine/swarm/>. [Accessed 8 11 2021].
- [30] OpenStack, “OpenStack,” 3 3 2022. [Online]. Available: <https://www.openstack.org/>.
- [31] XLAB, “xOpera SaaS,” [Online]. Available: <https://saas-xopera.xlab.si/ui/>. [Accessed 26 11 2021].
- [32] L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune and J. Wilkes, “Large-scale cluster management at Google with Borg,” in *Proceedings of the Tenth European Conference on Computer Systems*, 2012.

Appendix

In this appendix, the different scenarios for the IEM are represented in Gherkin, a cucumber specification format.

Feature: PIACERE Run Time

```
# The input of this scenario is detailed in the following
# https://git.code.tecnalia.com/piacere/private/t51-iem/iem/-
/blob/y2/openapi.json#/deployments/deploy_deployments_post
Scenario: Deploy a fresh project which comprises terraform, ansible,
and docker
Given a project bundle in the relevant IaC technologies (terraform,
ansible, docker-compose), the deployment id, and the required cloud
credentials
  When the user triggers the deployment
  Then the IEM is invoked
  And executes the stages of the bundle asynchronously
  And the user is notified that the deployment has been accepted

# The input of this scenario is detailed in the following
# https://git.code.tecnalia.com/piacere/private/t51-iem/iem/-
/blob/y2/openapi.json#/deployments/read_status_deployment_deployments_
_deployment_id_get
Scenario: Query the status of a running project
Given the deployment id of an already existing project
  When the user queries the status of the project
  Then the IEM is invoked
  And the user is notified of the status

# The input of this scenario is detailed in the following
# https://git.code.tecnalia.com/piacere/private/t51-iem/iem/-
/blob/y2/openapi.json#/deployments/undeploy_undeploy_post
Scenario: Undeploy a project
Given the deployment id of an already existing project and the
required cloud credentials
  When the user triggers the undeployment
  Then the IEM is invoked
  And tears down the entire deployment asynchronously
  And the user is notified that the undeployment has been accepted

# The input of this scenario is detailed in the following
# https://git.code.tecnalia.com/piacere/private/t51-iem/iem/-
/blob/y2/openapi.json#/deployments/read_status_deployment_deployments_
_deployment_id_get
Scenario: Query the status of an undeployed project
Given the deployment id of an undeployed project
  When the user queries the status of the project
  Then the IEM is invoked
  And the user is notified of the status
```

In it, the following scenarios are defined so other tools can assess how to utilize the IEM toolkit properly.

- **Deploy a fresh project:** this scenario represents a fresh multilingual IaC deployment in which the IEM can perform a fresh deployment of the different technologies that comprise the project.

- Query the status of a running project: the IEM can provide feedback on the various projects that have been executed, in conjunction with relevant messages that may be handy for problem resolution.
- Undeploy a project: in this scenario the IEM can tear down a previously deployed project that is no longer necessary or needs to be reassessed.
- Query the status of an undeployed project: this scenario provides feedback on a deployment that is no longer running on the infrastructure provider.

DRAFT