PIACERE

Deliverable D5.1

IaC execution platform prototype - v1– v1

Editor(s):	Josu Díaz de Arcaya
Responsible Partner:	TECNALIA
Status-Version:	V1.0
Date:	26.11.2021
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	30.11.2021

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

Abstract:	The main outcomes of Task 5.1 - 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	02.03.2021	First draft version	TECNALIA
V0.2	09.11.2021	Version delivered for internal review	All
V0.9	26.11.2021	Final version after internal review	TECNALIA
V1.0	29.11.2021	Ready for submission	TECNALIA
V1.1	18.11.2022	Error in references	TECNALIA

Table of contents

List of tables

List of figures

No se encuentran elementos de tabla de ilustraciones.

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

Executive Summary

This document contains the technical description of the IaC Execution Platform prototype, renamed as 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.

As for the future steps, this prototype will be hardened with authentication technologies so that its services cannot be misused by malicious third parties. Next, a persistence layer is going to be used to provide other components with metrics and data regarding past and present deployments. Finally, integration efforts will be conducted with other PIACERE components to interact successfully.

1 Introduction

This section offers a brief 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 servers the purpose of deployment and redeployment of the different scenarios that are developed by the PIACERE use cases. Being at month M12 of the project, the prototype described here is the first version, that is going to be integrated in the PIACERE framework version 1 at M15. 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 Key Result (KR) 10. 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 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. Section 3 revolves around the actual 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. In addition, a functional and technical description is offered for a better understanding of the prototype. Section 4 offers detailed information about the installation of the prototype and how to use it from scratch. Finally, the conclusions are presented in Section 5.

2 Tools aiding in the operationalization life cycle

2.1 Technologies needed by the use cases

It is of paramount importance to learn what technologies the use cases to be implemented within the PIACERE framework are using. Only this way it is possible to select the necessary technologies that will cover all the use case's requirements. The list below has been extracted from a careful examination of the Use Cases definition deliverable [1].

Technologies required:

- Virtualization technology including VMware vSphere, VMware vCenter, VMware vRealize (vOrchestration, vAutomation, vOperations, vBilling).
- DNS (bind).
- Certificate Authority for issuing SSL server certificates.
- Oracle Linux 8 based virtual machines.
- Docker + Swarm (within already configured virtual machines).
- Traefik (Cloud-Native Application Proxy).
- (Shared) object store CEPH (as container).
- SVN, GitLab (for sharing developed code and/or packages).
- Zabbix or another monitoring tool.
- AWS, Azure cloud providers.

2.2 Current state of art

In this section, a state of the art about the different IaC technologies that can be found in industry is offered. In addition, the applicability of these tools in the context of the PIACERE project alongside their shortcomings is offered.

2.2.1 Related existing technologies

2.2.1.1 Configuration Management

Configuration management is a process that ensures the consistency of a system over a period. It facilitates the automation of the life cycle of the configuration of the different infrastructural elements of an ecosystem and ensures concepts such as the maintainability, reproducibility, and traceability of the changes. In Table 1 a comparison of the most popular configuration management tools is offered.

Table 1 - Comparison of Configuration Management Tools.

Tool	First Release	Configuration Language	Language Style	Stars on GitHub
Ansible	2012	YAML	Procedural	47.2k
Chef	2009	Ruby DSL	Procedural	6.5k
Puppet	2005	Puppet DSL	Declarative	6.1k
SaltStack	2011	YAML	Declarative	11.6k
CFEngine	1993	DSL	Declarative	377

In terms of their first release, CFEngine [2] was the first to be conceived back in 1993 by Mark Burguess, and it was meant to be a tool to automate the management of a small group of workstations. The very first version was published as an internal report at the CERN computing conference, and it gained a lot of traction since it was able to abstract the different operative system flavors by using a common domain specific language (DSL). On the other hand, Ansible [3] was first released in 2012 by Michael DeHann, and it included a lot of the lessons learned

from other projects like Puppet [4]. It aspired to be a system that was easy to learn, consistent, and imposed minimum dependencies on the managed systems. All the tools being analyzed offer their respective configuration language. While Ansible and SaltStack [5] promote the use of YAML, CFEngine, Puppet and Chef [6] require the use of their respective DSL. In terms of the language style there are two different styles that are utilized by the tools under analysis: procedural and declarative. Procedural languages promote the definition of the whole process as a series of steps that will be executed, how the process will be executed is defined in detail. On the other hand, declarative languages define what needs to be done, and how the system will look at the end of the process, but how to reach that point is up to the tool itself. In this sense, Ansible and Chef adopted a procedural language style, whereas Puppet, SaltStack and CFEngine adopted a declarative language style. Finally, in order to measure the popularity of each of the tools, we have opted for an objective metric which is the number of stars on GitHub. According to this, Ansible is the most popular one in this site by a large margin, more than three times the number of stars of the second one, SaltStack. Chef and Puppet are the third and fourth in this metric with 6.5k and 6.1k stars respectively.

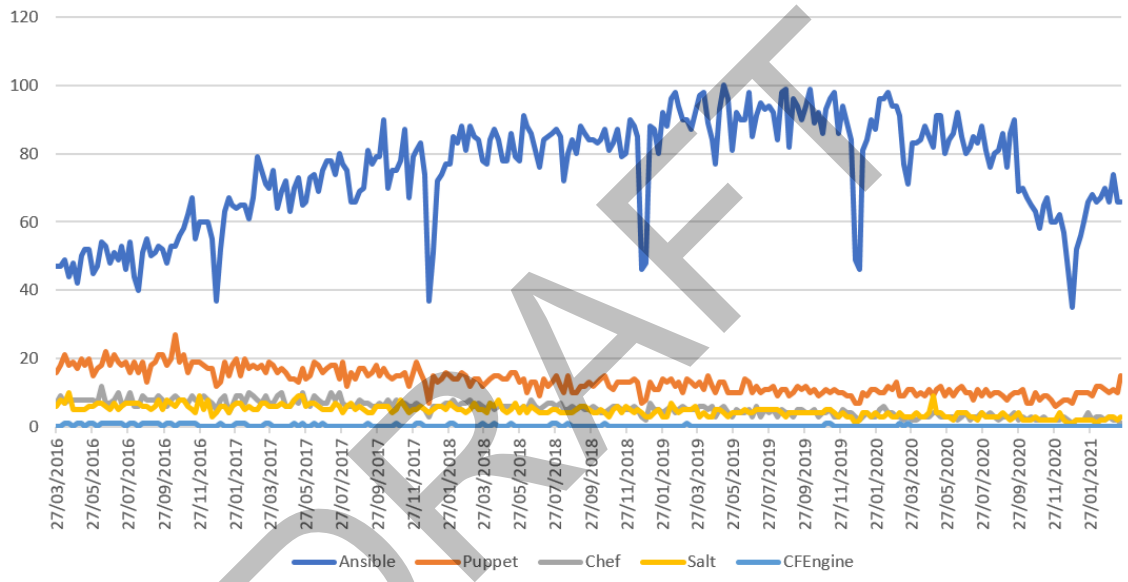

Figure 1 - Popularity of Configuration Management Tools according to Google Trends over time.

Yet another way of evaluating the popularity of these tools is with Google Trends, which shows the frequency of search terms in google in a given period. The results (see Figure 1) are somehow related with the popularity on GitHub. In this sense, Ansible keeps being the most popular tool by a generous margin and has been so for the last five years. The second most popular tool according to Google Trends is Puppet, which differs from the previous approach, in which SaltStack was the second most popular tool. The rest of the tools seem consistent with the analysis made before.

2.2.1.2 Infrastructure Provisioning

The Infrastructure provisioning is the process of setting up and IT infrastructure, referring to the steps required to manage access to data and resources and make them available to users and systems.

When the term "provisioning" is used, it can mean many different types of provisioning, such as server provisioning, network provisioning, user provisioning, service provisioning, and more.

The following products provide the required functionalities to implement this and to monitor this process:

CloudFormation [7] is an infrastructure templating, or Infrastructure as Code (IaC) service provided free of charge exclusively by AWS. It employs an active management strategy, performing automated scaling and healing. As it might be expected, it has extensive support for not only EC2 infrastructure, but a wide variety of AWS services. It is provided free of charge. It is a closed source.

Terraform [8] is an infrastructure as code (IaC) tool that allows to build, change, and version infrastructure safely and efficiently. This includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc. It has historically been a Command Line Interface (CLI) only tool, targeting all major public clouds. Recently, it has been offered as a SaaS product as well. It employs a passive management strategy, performing tasks based on user requests. The enterprise version has support for multi-tenancy and deployment policies. It is an open-source product, governed by HashiCorp.

Openstack – Heat [9] (the main project in the OpenStack Orchestration program) implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code. A native Heat template format is evolving but Heat also aspires to provide compatibility with the AWS CloudFormation template format, so that many existing CloudFormation templates can be launched on OpenStack. Heat provides both an OpenStack-native REST API and a CloudFormation-compatible Query API.

A Heat template describes the infrastructure for a cloud application in a text file that is readable and writable by humans and can be checked into version control. That file must be enabled for writing and it should be possible to check the difference between the versions. The keyword is the resource group where the user can define infrastructure: volumes, servers, floating IP addresses, volumes, security groups, users etc.

The relationship between the resources in the template can be specified and it is enabled to call out any of OpenStack API to create your whole infrastructure in one run. The Heats architecture comprises several Python applications listed below:

- **heat**: this tool is a CLI which communicates with the heat-api to execute AWS CloudFormation APIs. Of course, this is not required, developers could also use the Heat APIs directly.
- **heat-api**: this component provides an OpenStack-native REST API that processes API requests by sending them to the heat-engine over RPC.
- **heat-api-cfn**: this component provides an AWS-style Query API that is compatible with AWS CloudFormation and processes API requests by sending them to the heat-engine over RPC.
- **heat-engine**: it does the main work of orchestrating the launch of templates and providing events back to the API consumer.

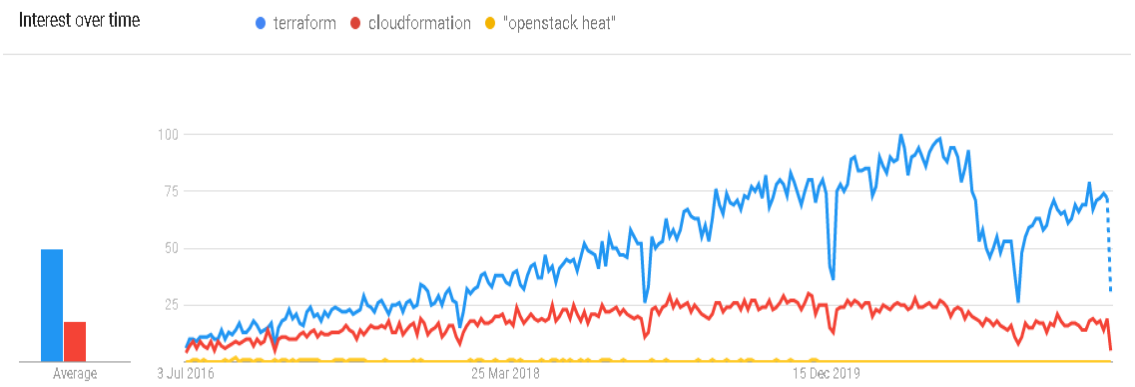

Figure 2 – Popularity of Provisioning Tools according to Google Trends over the last 5 years.

Figure 2 provides and overview of the evolution of the popularity of these tools over the last five years. It can be seen that Terraform is significantly more popular than its counterparts.

2.2.1.3 Container Orchestration

Container based virtualization is a method in which a virtualization layer operates on top of the operative system enabling the coexistence of several user space instances. These instances are what it is called containers. It differs from other methods of virtualization - such as the traditional based on hypervisors - in which the full set of hardware components is not virtualized. The first attempt to such methods of virtualization dates back to 1982, when Bill Joy developed chroot. In chroot a security zone is created from which the jailed user cannot escape. More recently, in 2008 the first version of Linux Containers was released based on the cgroups functionality.

However, it was not until 2013 that docker containers were launched and became the *de facto* technology for containerizing applications in Linux, Windows, and macOS. It utilized kernel facilities (mainly cgroups and namespaces) for providing resource isolation (e.g. CPU, memory, I/O) and provides means for the operationalization of software applications in production environments, while minimizing the difficulties arising from deploying applications in different environments. In 2014 the CoreOS team launched rkt as an alternative to docker containers. Later in 2018 CoreOS was acquired by Red Hat and released Podman, which is the main container virtualization tool in Red Hat 8.

Table 2 - Comparison of container orchestration tools.

Tool	First Release	Stars on GitHub	Language	Container Runtime
Docker Swarm	2013	-	YAML	docker
Kubernetes	2014	27.4K	YAML	Containerd, docker, CRI-O
Marathon	2014	4k	JSON	UCR, docker
YARN	-	11.5k[1]	JSON	runC, docker

The table above offers an overview of some of the most popular container orchestration tools that can be utilized nowadays. The very same company that released the docker containerization technology released docker swarm in 2013. Docker swarm [10] offers the most common characteristics that can be expected of a container orchestration tool such as cluster management, scaling, service discovery, load balancing, rolling updates, among others. It has

[1] It is shared with the wider project Apache HADOOP.

become an extremely popular alternative to other container orchestration tools in on premise deployments due to being very easy to setup and maintain. However, its functionality is more limited than other alternatives such as Kubernetes.

Kubernetes [11] is probably the most popular and feature rich container orchestration available, the project was developed from the Google's Borg [12] project. The main benefits of Kubernetes are that is backed up by a huge community, is very feature rich, and has been accepted as the *de facto* container orchestration of the major cloud providers. This makes Kubernetes an excellent choice for the containerization of applications, since it diminishes the chance of suffering vendor lock-in due to its wide adoption in the market. Its main drawbacks are that the installation on premises is quite complex and not particularly easy to manage, and the learning curve required is more steep than other options such as docker swarm. It is common to find private companies that help with the management of Kubernetes. For instance, Rancher [13] is a technology for organizing and managing Kubernetes clusters. Major cloud providers such as AWS, GCP, and Azure also facilitate the use of Kubernetes as a Service.

Next, Marathon [14] is the container orchestration tool for the Mesos based ecosystems. It is the built-in container orchestration of the Datacenter Operating System (DC/OS), which is an open-source operative system that can manage multiple machines both in the cloud and on-premises. It offers networking, service discovery and resource management.

Finally, the HADOOP ecosystem, which introduced the HDFS distributed file system and allowed exploiting distributed datasets using YARN due to its data locality features. YARN [15] introduced the ability to orchestrate containers, and it has become particularly useful for organizations already familiar with the HADOOP ecosystem that already have clusters using it. It provides a straightforward way to incorporate container based microservice architectures and offers a single point to manage the resource of the cluster in YARN.

In terms of popularity, the number of stars on GitHub is a common approach to measure it. Using this metric, Kubernetes is the most popular tool with over 27k stars, followed by Apache YARN with 11.5k. However, given that the term YARN refers to a wider ecosystem, this metric does not reflect accurately the popularity of the container orchestrator.

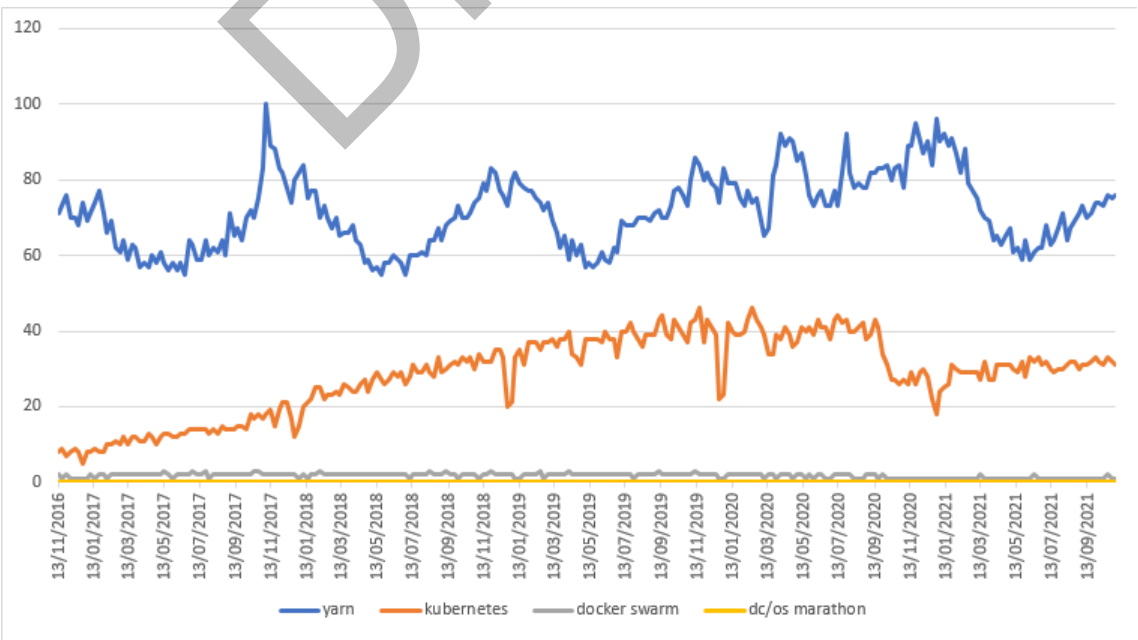

Figure 3 – Popularity of Container Orchestrator Tools according to Google Trends over the last 5 years.

Secondly, Google Trends is a common tool for evaluating the popularity of a tool based on its searches. Using this metric (see Figure 3 above), YARN is the most popular of the tools mentioned above. However, it is a very general term and does not only refer to the container orchestration part. Kubernetes is second, even though its popularity has slightly declined in 2021, it is by far the most widely adopted option. Finally, docker swarm and DC/OS' Marathon are the third and last.

2.2.1.4 Deployment Orchestration

This term refers to different to the execution of full workflows of low-level operations such as the provisioning of resources, the configuration and installation of components, the connection between the components and the management of the dependencies, amongst others. These tools are able to utilize some of the different tools described above from a common interface. In particular, they approach the problem in hand as a whole, leveraging the benefits of configuration management tools, infrastructure provisioning, and application orchestration, while minimizing the drawbacks of them when possible. Summarizing, they offer a unified point to manage the ecosystem.

In this sense, Apache Brooklyn's [16] initial release dates back to 2012, it defines itself as a framework for modeling, monitoring, and managing applications through autonomic blueprints. It builds on the foundations established by other tools such as configuration management tools, and infrastructure provisioning tools and provides an integration framework to reconcile their use. Its main benefit is providing a single tool to manage provisioning, application deployment, and monitoring, which understands the dependencies between the components. It is built with the focus of application agility, including portability across non-cloud, cloud, and PaaS.

Next, Spinnaker [17] is a cloud agnostic continuous delivery platform that focuses on the operationalization of software components with high velocity and confidence. It utilizes pipelines comprised of a series of stages. These pipelines aspire to be flexible, consistent and repeatable.

Another interesting orchestrator is xOpera [18], which derives from the Opera orchestrator available as a Python library, which is a light-weight, open-source and state-aware orchestrator based on Ansible and TOSCA Simple Profile in YAML v1.3. Initially, this orchestrator was only CLI based, but in later versions it was expanded with API accessibility and, simultaneously, an orchestrator as a service edition called xOpera SaaS [19] has emerged. The latter enables integration with other authorization tools, handling secrets, providing multi-tenancy (sharing deployment projects with co-workers), accessibility through a GUI and more.

Finally, Cloudify [20] is an opensource cloud orchestration tool, that focuses on the automation of the operationalization life cycle of applications. Its key features are the following: Everything-as-a-Code, Orchestrator of Orchestrators, Native CI/CD support, and Operability, amongst others.

2.2.2 Orchestration in H2020 projects

There are a few Horizon 2020 projects that dive into the concept of orchestration, the SERRANO [21] 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 [22] project tackles the underlying issues of technological developments in the field of virtual reality, augmented realty and holography by utilizing new cloud computing architectures and the continuous and autonomous orchestration of computing and network resources. The RADON [23] 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 [24] 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 provides 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.

2.2.3 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, where 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 require less configuration at the beginning. This is a big advantage, since the PIACERE deployments are 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.

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.

2.2.4 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.

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 [1] 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 3.1 the requirements that have been identified by the project for the IEM to address are presented. Next, Section 3.2 showcases the functional description and how the IEM fits in the overall PIACERE architecture. Section 3.3 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 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 [26] 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 3 - Requirements that need to be addressed by the IEM and its current fulfilment status.

Req ID	Description	Status	Requirement Coverage
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.	Not yet covered	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 and enable an appropriate way to query it.	Not yet covered	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-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.	Partially 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.

3.2 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.

3.2.1 Fitting into overall PIACERE Architecture

The IEM is the component of the PIACERE architecture that executes the DevOps modelling Language (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 4. 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.

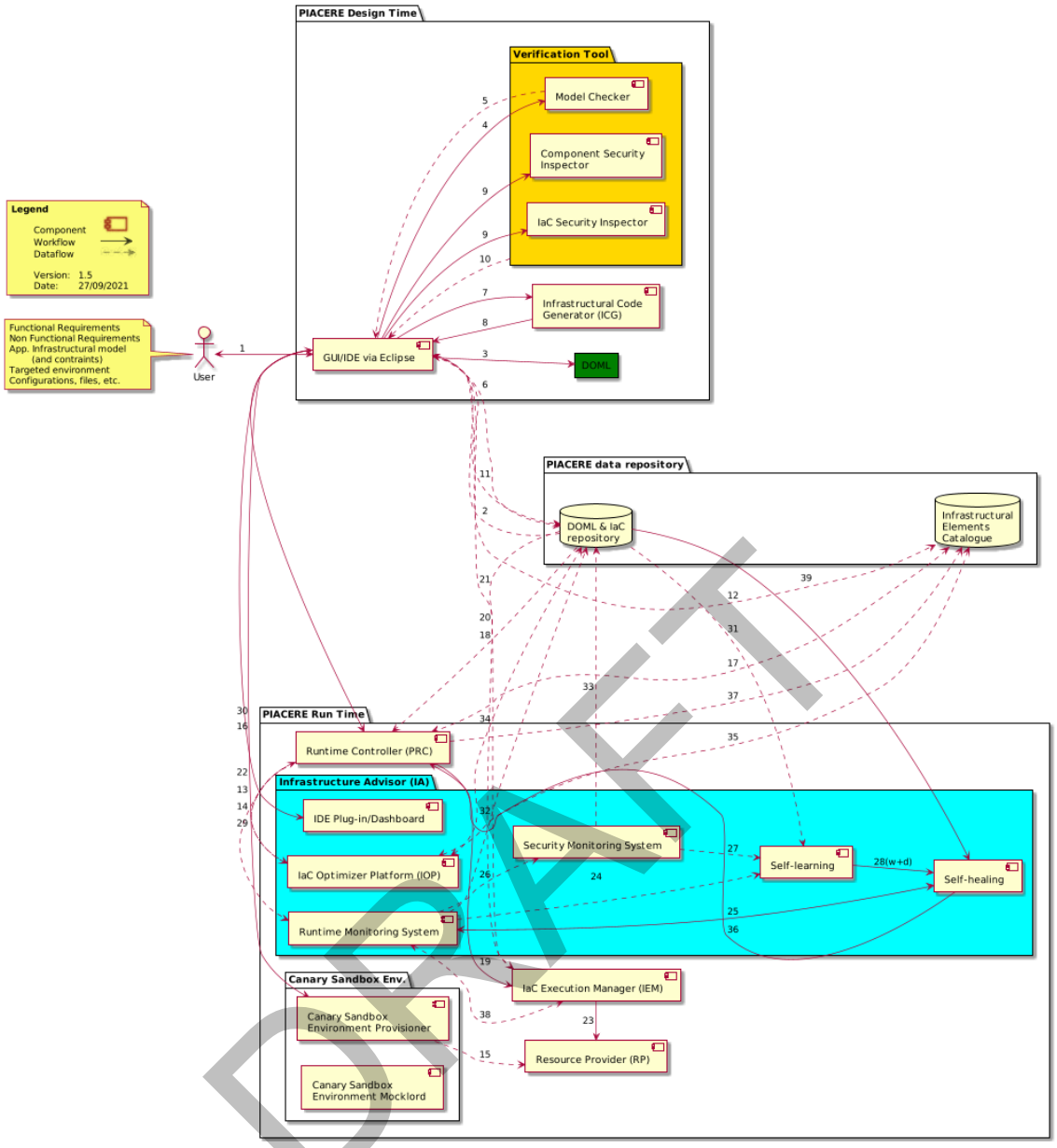

Figure 4 – PIACERE Runtime Diagram (version 1.5)

3.3 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.

3.3.1 Prototype architecture

The architecture components that comprise the entirety of the IEM component are depicted in the following Figure 5. This 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.

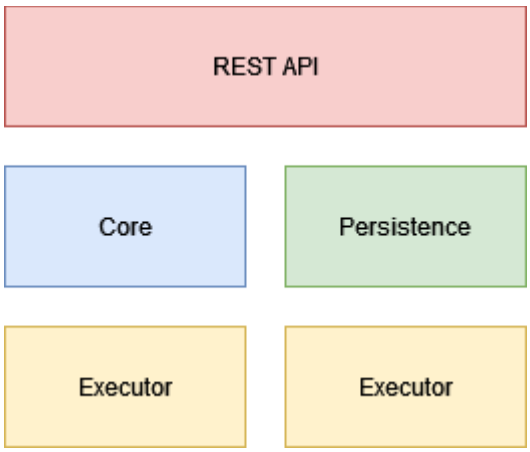

Figure 5 - Architecture of the different elements comprising the IEM Execution Manager

3.3.2 Components 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 will be 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, and ii) Terraform for the provisioning of the infrastructural elements required for the deployments to be executed successfully.

3.3.3 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 will be undergone during the coming year of the project but is depicted in this deliverable for a better understanding of the component duties.

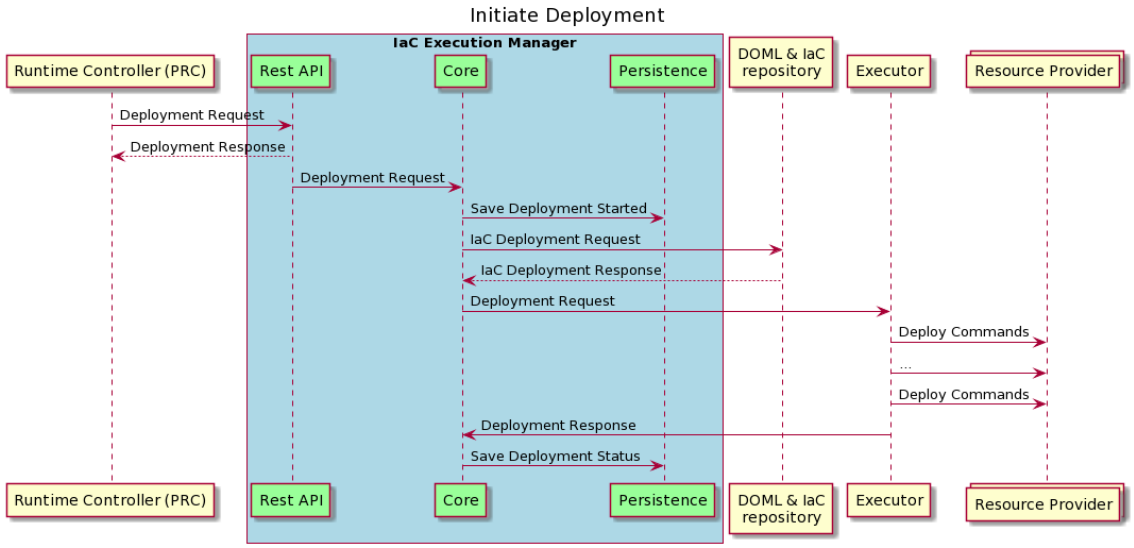

Figure 6 - Sequence Diagram for the process of kicking off a deployment

The Figure 6 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 7). The security of this communication will be 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.

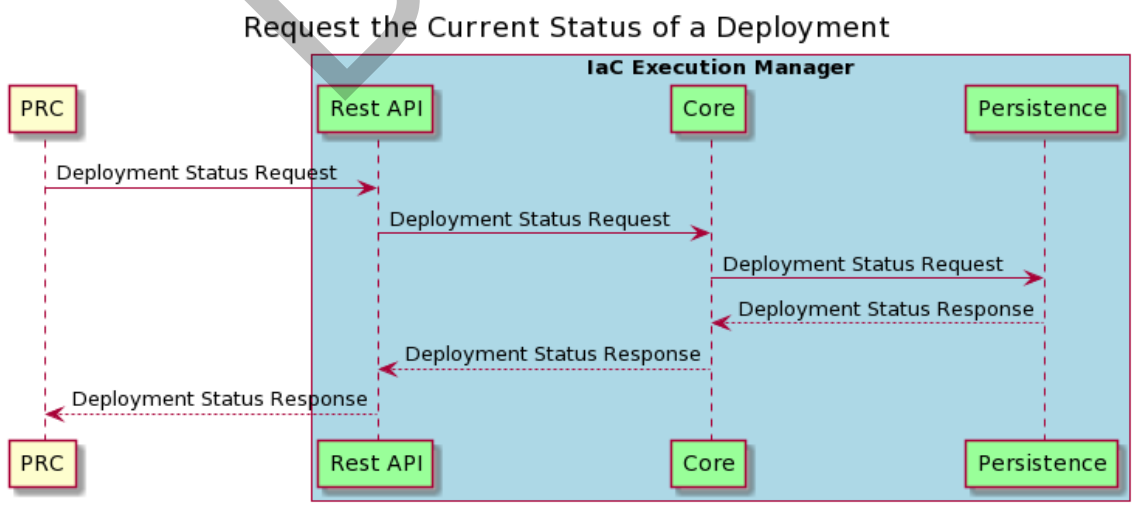

Figure 7 – Sequence diagram of the process of requesting the status of a given deployment

The Figure 7 above exemplifies the process of querying the status of a given deployment. This functionality is not yet implemented as part of the prototype and will be undergone in 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.

3.3.4 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 will be undergone during the second year of the project. The persistence layer will be implemented using PostgreSQL, and it will contain 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.

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

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 8 – 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.

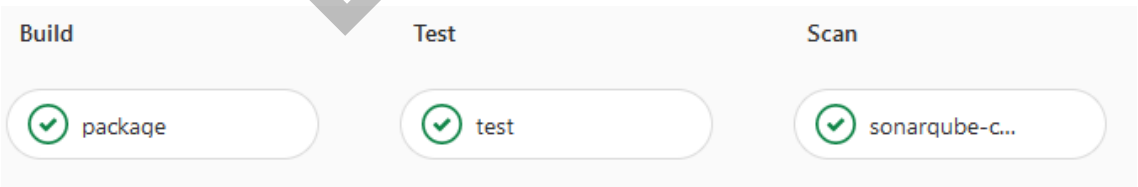

Figure 9 – Continuous Integration and Deployment pipeline for the IEM project

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 10).

Figure 10 – 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 Tecnalia'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.

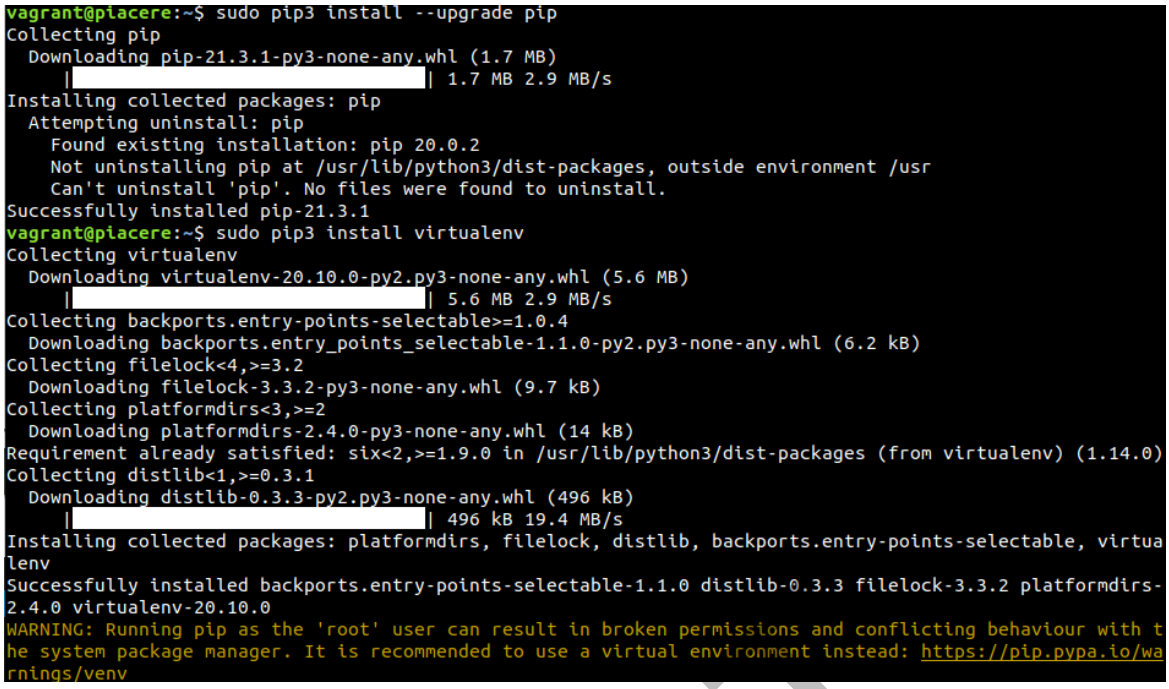

Figure 11 - Excerpt showing the installation steps for virtualenv in a linux box

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

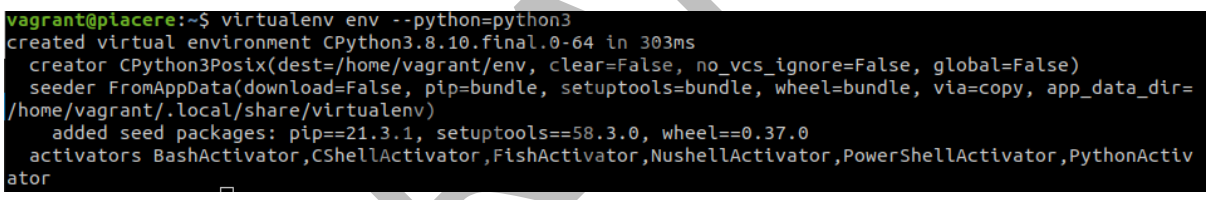

Figure 12 - Excerpt showing the creation of a virtual environment

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

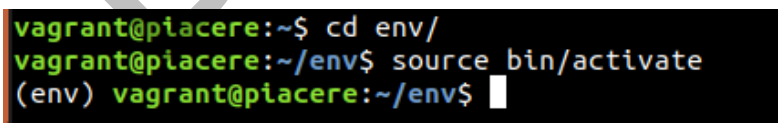

Figure 13 - Excerpt showing the activation of the freshly created virtual environment

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.

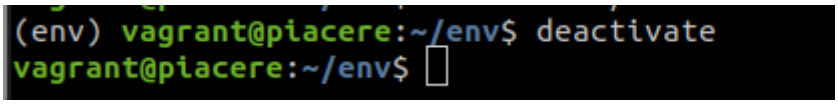

Figure 14 - Excerpt showing how to finalize or deactivate a virtual environment

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.

```
$ cat requirements.txt
fastapi[all]==0.64.0
ansible==2.9.1
docker==5.0.0
GitPython==3.1.18
$ pip3 install -r requirements.txt
```

Figure 15 - 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 documentation[2] 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.

```
$ 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 16 - 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.

```
$ 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 17 - 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. For this

[2] https://learn.hashicorp.com/tutorials/Terraform/install-cli

prototype, Terraform is not yet covered within the containerized version. It will be covered in future versions.

```
FROM python:3.9.5

COPY iem/src/resources/ansible.cfg /etc/ansible/ansible.cfg
COPY iem/requirements.txt /tmp/requirements.txt
COPY iem/ /opt/iem

RUN pip3 install -r /tmp/requirements.txt
WORKDIR /opt/iem
CMD /usr/local/bin/uvicorn main:app --reload --host 0.0.0.0
EXPOSE 8000
```

Figure 18 - 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 4 - Excerpt showing the built and execution of the IEM container

```
$ docker build . -t iem

$ docker run -p 8000:8000 iem
```

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.

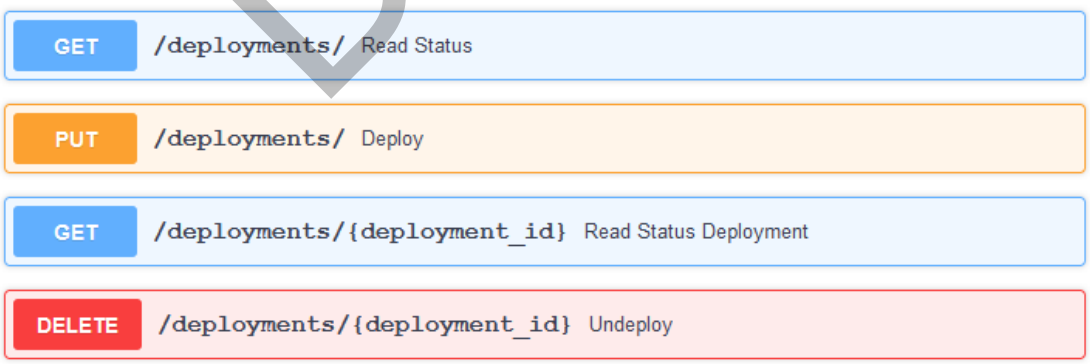

Figure 19 - OpenAPI specification for the interaction with the IEM component

The Figure 19 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.
- *PUT /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 20 - 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 paremeter.
- *DELETE /deployments/{deployment_id}*: the deployment specified in the deployment_id path parameter will be torn down.

Please keep in mind that for the first prototype only the *PUT /deployments/* endpoint is fully functional. The rest of the endpoints are usable but provide no real functionality. The reasoning behind their existence at this early stage is to 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 21 - 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.

```python
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 22 - 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 23 - 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 5 - 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

Licensing information not yet included.

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

The source code will be available on the public git repository and accessible through the project's website https://www.piacere-project.eu/. At the time of writing this deliverable, the source code is provided under request through an email to the address appearing on the website (https://www.piacere-project.eu/) in the footer under "Contact Us".

5 Conclusions

This document revolves around the first 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 first 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.

Future iterations of this prototype will include the securitization of the different endpoints to be used by the IEM with token-based technologies. In addition, a persistence layer will be 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., Terrraform 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. TOSCA) if desired.

6 References

[1] P. Consortium, "D7.1 PIACERE use case definition," 2021.

[2] CFEngine, "CFEngine," [Online]. Available: https://cfengine.com/.

[3] Red Hat, "Red Hat Ansible," [Online]. Available: https://www.ansible.com/.

[4] puppet, "puppet," [Online]. Available: https://puppet.com/.

[5] SaltStack Enterprise, "Salt Project," [Online]. Available: https://saltproject.io/.

[6] CHEF, "CHEF," [Online]. Available: https://www.chef.io/products/chef-infra.

[7] AWS, "AWS CloudFormation," [Online]. Available: https://aws.amazon.com/es/cloudformation/. [Accessed 8 11 2021].

[8] HashiCorp, [Online]. Available: https://www.terraform.io/. [Accessed 8 11 2021].

[9] OpenStack, "OpenStack," [Online]. Available: https://www.openstack.org/. [Accessed 8 11 2021].

[10] Docker Inc, "Swarm mode overview," [Online]. Available: https://docs.docker.com/engine/swarm/. [Accessed 8 11 2021].

[11] The Linux Foundation, "Production-Grade Container Orchestration," [Online]. Available: https://kubernetes.io/. [Accessed 8 11 2021].

[12] 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.

[13] Rancher Labs, "Rancher," [Online]. Available: https://rancher.com/. [Accessed 26 11 2021].

[14] Mesosphere Inc., "Marathon - A container orchestration platform form Mesos and DC/OS," [Online]. Available: https://mesosphere.github.io/marathon/. [Accessed 8 11 2021].

[15] Apache Hadoop, "Apache Hadoop YARN," [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html. [Accessed 8 11 2021].

[16] Apache Brooklyn, [Online]. Available: https://brooklyn.apache.org/. [Accessed 8 11 2021].

[17] Spinnaker, "Spinnaker," [Online]. Available: https://spinnaker.io/. [Accessed 8 11 2021].

[18] XLAB, [Online]. Available: https://github.com/xlab-si/xopera-opera. [Accessed 26 11 2021].

[19] XLAB, "xOpera SaaS," [Online]. Available: https://saas-xopera.xlab.si/ui/. [Accessed 26 11 2021].

[20] Cloudify Platform Ltd., "Multi Cloud Orchestration: Turn Glue Code Into Certified Environments," [Online]. Available: https://cloudify.co/. [Accessed 8 11 2021].

[21] 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].

[22] CHARITY, "Cloud for Holography and Cross Reality," [Online]. Available: https://cordis.europa.eu/project/id/101016509. [Accessed 9 11 2021].

[23] RADON, "Rational decomposition and orchestration for serverless computing," [Online]. Available: https://cordis.europa.eu/project/id/825040. [Accessed 9 11 2021].

[24] ELEGANT, "Secure and Seamless Edge-to-Cloud Analytics," [Online]. Available: https://cordis.europa.eu/project/id/957286. [Accessed 9 11 2021].

[25] DICE, "Data Infrastructure Capacity for EOSC," [Online]. Available: https://cordis.europa.eu/project/id/101017207. [Accessed 26 11 2021].

[26] P. Consortium, «D2.1 PIACERE DevSecOps Framework Requirements specification, architecture and integration strategy,» 2021.