



PIACERE

Deliverable D5.6

Canary Sandbox Environment prototype – v3

Editor(s):	Radosław Piliszek (7bulls)
Responsible Partner:	7bulls
Status-Version:	Final – v1.0
Date:	31.05.2023
Distribution level (CO, PU):	PU

Project Number:	101000162
Project Title:	PIACERE

Title of Deliverable:	Canary Sandbox Environment prototype
Due Date of Delivery to the EC	31.05.2023

Workpackage responsible for the Deliverable:	WP5 - Package, release and configure IaC
Editor(s):	Radosław Piliszek (7bulls)
Contributor(s):	Radosław Piliszek (7bulls)
Reviewer(s):	Iñaki Etxaniz (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	Mandatory: WP3 (IDE), WP5 (IEM) Recommended: WP2 (Architecture, Integration)

Abstract:	<p>The final outcomes of Task 5.2 - PIACERE Canary environment for IaC behaviour testing and simulation - are presented in this deliverable.</p> <p>This deliverable corresponds to Key Result 8 – Canary Sandbox Environment.</p> <p>This is v3, the final version, of the deliverables D5.4/5.5 and reports the final progress, lessons learnt and outlook to the future.</p>
Keyword List:	Canary environment, sandbox, dynamic testing
Licensing information:	<p>This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)</p> <p>http://creativecommons.org/licenses/by-sa/3.0/</p>
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	08.05.2023	First draft version	Radosław Piliszek (7bulls)
v0.2	15.05.2023	New v3 content	Radosław Piliszek (7bulls)
v0.3	26.05.2023	Amendments after the internal review	Radosław Piliszek (7bulls)
v0.4	29.05.2023	Update of the WP2 diagram.	Radosław Piliszek (7bulls)
v1.0	30.05.2023	Final quality check. Ready for submission.	Juncal Alonso (Tecnalia)

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 KR 8 overview.....	9
2.1 Changes in v3	9
2.2 Functional description and requirements coverage	9
2.2.1 Canary Sandbox Environment Provisioner - CSEP.....	10
2.2.2 Canary Sandbox Environment Mocklord - CSEM	11
2.3 Main innovations.....	12
3 Overview of preliminary experiments.....	13
4 Lessons learnt and outlook to the future.....	14
5 Conclusions	15
6 References.....	16
APPENDIX: Implementation, delivery and usage	17
1 Implementation.....	17
1.1 Fitting into overall PIACERE Architecture.....	17
1.2 Technical description	17
1.2.1 Canary Sandbox Environment Provisioner - CSEP.....	17
1.2.2 Canary Sandbox Environment Mocklord – CSEM	20
2 Delivery and usage	23
2.1 Canary Sandbox Environment Provisioner - CSEP.....	23
2.1.1 Package information	23
2.1.2 Installation instructions.....	24
2.1.3 User Manual	24
2.1.4 Licensing information.....	27
2.1.5 Download	27
2.2 Canary Sandbox Environment Mocklord - CSEM	27
2.2.1 Package information	27
2.2.2 Installation instructions.....	27
2.2.3 User Manual	28
2.2.4 Licensing information.....	29
2.2.5 Download	29

List of tables

TABLE 1 - REQUIREMENTS TO BE ADDRESSED BY THE CANARY SANDBOX ENVIRONMENT TOOLING AND THEIR STATUS IN THIS RELEASE. 9

List of figures	
FIGURE 1 - CSEP MAIN SEQUENCE DIAGRAM	10
FIGURE 2 – EXAMPLE CSEM SEQUENCE DIAGRAM	12
FIGURE 3 - PIACERE RUNTIME DIAGRAM FOR DEPLOYMENT (FROM D2.2).....	17
FIGURE 4 - CSEP ARCHITECTURE DIAGRAM	18
FIGURE 5 - API ENDPOINTS AS SHOWN BY SWAGGER UI AT /DOCS.....	25
FIGURE 6 - AN EXAMPLE API REQUEST (POST NEW DUMMY DEPLOYMENT)	26
FIGURE 7 - AN EXAMPLE API RESPONSE (POST NEW DUMMY DEPLOYMENT).....	26

DRAFT

Terms and abbreviations

AWS	Amazon Web Services (the most popular public cloud provider, used also by the PIACERE use case partners)
CRP	Canary Resource Provider (a resource provider created by CSE tooling)
CSE	Canary Sandbox Environment (an umbrella term for this series of deliverables and the produced tools)
CSEM	CSE Mocklord (one of the proposed tools, related to lightweight, simulated approach)
CSEP	CSE Provisioner (one of the proposed tools, responsible for target environment provisioning)
CSP	Cloud Service Provider
DevOps	Development and Operations
DevSecOps	Development, Security and Operations
DoA	Description of Action
IaC	Infrastructure as Code
IEM	IaC Execution Manager (one of the tools built in the PIACERE project, delivered as part of Task 5.1)
KR	Key Result
KR8	Key Result 8 (which this deliverable is about)
KR13	Key Result 13 (the integrated PIACERE framework, i.e., all tools of PIACERE working together, CSE tools being a part of them)
TSR	Technical Specification Report (e.g., this document)

Executive Summary

This document is the Technical Specification Report (TSR) of the third and final version of the Canary Sandbox Environment (CSE) – Key Result (KR) 8 – tooling prototype developed in the PIACERE project. The CSE KR provides the tools necessary for ensuring the availability of platforms for testing of user's Infrastructure as Code (IaC) in a sandbox way. It is the result of the effort of Task 5.2 in Work Package 5 of the PIACERE project.

As this (D5.6) is version 3 (final) of "Canary Sandbox Environment prototype" TSR, it's based on both previous versions of the same-named deliverable (v1 in D5.4 and v2 in D5.5). The main content has been refocused on the final outcomes, including the latest work results, lessons learnt and outlook to the future. To make this document self-contained, the majority of the previously-existing content has been preserved in the appendix, updated where applicable to reflect the current state of the tooling.

The main content starts with an introduction and then moves onto discussing the latest version of the delivered KR, its functionalities and innovations. In comparison to the previous versions of this deliverable, there are two new sections on the experiments and lessons learnt with outlook to the future. Finally, the conclusions complete the TSR by summarising the effort.

The tooling reported in this deliverable, in this third revision, retains the existing functionalities from previous versions. First of all, it enables users to provision OpenStack which can then be utilised by the IaC Execution Manager (IEM). Additionally, in v2, PIACERE users were offered an AWS-mocking solution based on an existing project but enhanced beyond the original capabilities and packaged for easy use in the PIACERE project. In v3, the focus was on polishing these tools, e.g., the desired undeployment functionality for OpenStack has been delivered.

The provisioner's usability is high and its support depends on keeping in sync with the upstream OpenStack community. On the other hand, the Mocklord is more problematic because of a relatively high support cost to effect ratio and the related limited scope and applicability.

While this is the last formal version of this deliverable, it is likely that the tooling itself and its separate description will continue to evolve beyond the duration of Work Package 5.

1 Introduction

1.1 About this deliverable

This document is the Technical Specification Report (TSR) of the third (final) version of Canary Sandbox Environment (CSE) tooling prototype developed in the PIACERE project. The goal of the TSR is to put the proposed software in context, define its architecture and the means of use. The CSE tooling is Key Result 8 (KR8) in the PIACERE project developed as part of Task 5.2 in Work Package 5.

In this final version, there are two proposed tools being described: Canary Sandbox Environment Provisioner (CSEP) and Canary Sandbox Environment Mocklord (CSEM). CSEP is used to provision Canary Resource Providers (CRPs); OpenStack is offered as an example, but the implementation is extensible. CSEM, on the other hand, focuses on mocking cloud providers, such as AWS in this prototype, and avoiding costs related to provisioning of real infrastructure. The CSEM's mocked AWS can also act as a CRP. CRPs can be used by other PIACERE tooling, most notably the IaC Execution Manager (IEM), to realise the dynamic testing of Infrastructure as Code (IaC), as they are meant to replace the actual, production resource providers. The details on these functionalities have been provided in Section 2.2.

1.2 Document structure

The TSR for this third (final) version has been re-organised project-wide slightly in comparison with the v1 and v2 deliverables. It is organised in 6 main sections. The 1st section is this introduction containing the information on the TSR itself. The 2nd section describes the Key Result (KR). The 3rd section provides an overview of the preliminary experiments. The 4th section discusses lessons learnt and outlook to the future. The 5th section concludes the document. The 6th section provides a list of references. The document ends with an appendix including updated content from v2 of this deliverable, retaining much of the practical details regarding the technical aspects of the implementation and operation of the provided tooling, including the user guide.

2 KR 8-CSE overview

KR 8, the Canary Sandbox Environment (CSE), focuses on providing tooling to enable sandbox-like testing of IaC before deploying it in production (thus canary). It can be used with and without the rest of the PIACERE framework. It has been successfully used to provide a testbed for the rest of the PIACERE framework.

2.1 Changes in v3

The tools were developed to satisfy the posed requirements in v2. In v3, we focused on polishing the interactions. Most notably, the requested undeployment functionality has been added to CSEP (CSE Provisioner) that enables the user to clean up entirely the deployed OpenStack and possibly start anew on the same set of resources used originally (or repurpose them).

2.2 Functional description and requirements coverage

The proposed prototypes are offering two approaches to the CSE: a real (non-simulated) Canary Resource Provider (CRP) and a simulated one. Any CRP is to be used by the IaC Execution Manager (IEM) – another tool from the PIACERE project – as a target when running IaC, replacing the actual, production resource provider. That said, the CRPs are general enough and can be used with the same tools actual, production providers would be used, such as IaC tooling (e.g., Terraform [1] or Ansible [2]) or dedicated clients.

Depending on the CRP variant, the scope and characteristics of testing differs. Real providers require resources and allow to complete all steps of deployment as long as the supporting infrastructure (beneath the created CRP) is sufficient. The assumption is that the user is able to provide the hardware (e.g., because they have bare metal or virtual machines, either on premise or elsewhere – the CSE is agnostic to that). On the other hand, the simulated variant does not consume resources but does not allow further steps other than provisioning of the infrastructure elements.

The set of supported CRPs is: OpenStack (for real [non-simulated] actions) and Canary Sandbox Environment Mocklord (CSEM; for simulation). However, the tooling is extensible and could support other environments in the future.

As part of Work Package 2 efforts¹, certain requirements against CSE have been gathered and they are presented along with their status in Table 1 - Requirements to be addressed by the Canary Sandbox Environment tooling and their status in this release. All the original requirements were completed in v2 while v3 focused solely on improving the tools based on feedback.

Table 1 - Requirements to be addressed by the Canary Sandbox Environment tooling and their status in this release.

Req ID	Description	Status	Comment
REQ33 / WP5.2-REQ1	CSE to provide a viable alternative target for IaC executors to run against, i.e. usable by the IaC Execution Manager (IEM).	Implemented	No change.
REQ34 / WP5.2-REQ2	CSE to keep track of and allow querying of the deployment state to allow comparison against the expected one.	Implemented	No change.
REQ37 / WP5.2-REQ3	CSE to have a simulated mode limited to provisioning.	Implemented	No change.

¹ Reported in D2.1 and D2.2.

REQ38 / WP5.2-REQ4	CSE to have a "real" mode where resources are really provided and can be used for configuration and other further steps.	Implemented	No change.
REQ39 / WP5.2-REQ5	CSE to enable extensibility (documented way): adding new mocked services, adding new "real" deployments.	Implemented	No change.

The functionality and purpose of the two provided tools are explained in the following two subsections.

2.2.1 Canary Sandbox Environment Provisioner - CSEP

The role of the Canary Sandbox Environment Provisioner (CSEP) is to create the desired Canary Resource Provider (CRP). This may entail provisioning and configuring new systems to provide the expected platform. The main sequence diagram is presented in Figure 1 - CSEP Main Sequence Diagram. It has not changed since v1.

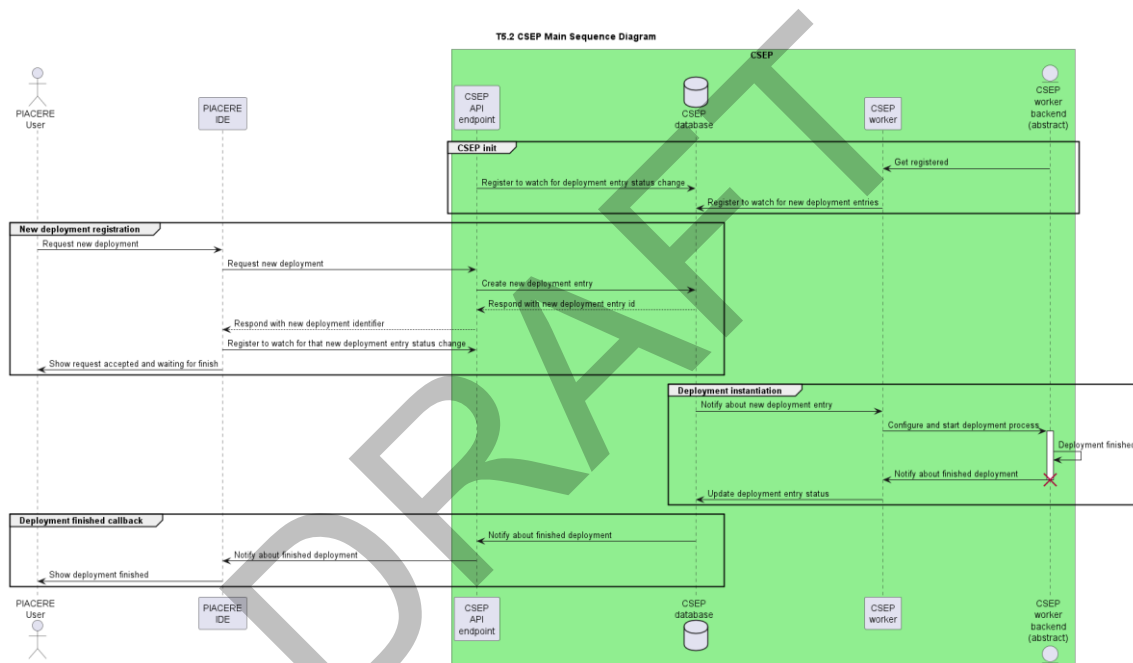


Figure 1 - CSEP Main Sequence Diagram

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

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

The alternative and complementary flows involve destroying the deployment (when the flow of actions is analogous to creation), listing deployments, and getting details about a particular deployment.

2.2.2 Canary Sandbox Environment Mocklord - CSEM

The role of Canary Sandbox Environment Mocklord (CSEM) is to simulate an existing resource provider so that the user can easily test interactions against it. The current prototype targets a subset of AWS APIs. CSEM has much lower cost compared to real (non-simulated) resource providers. Due to simulation, this variant of Canary Resource Provider allows only the IaC steps targeted directly against the resource provider to succeed as no real resources will be provided and, thus, no actions can target them, e.g., it is not possible to connect to the “deployed” virtual machine as there is none to target.

The goal with CSEM is to evaluate the applicability of such mocked testing of IaC code in real life. To this end, it has been validated with an example from one of the use case partners of PIACERE. However, as this is a very new addition in the PIACERE family, its testing is still in the works. Overall, this component tries to behave like the actual AWS API without consulting it. Another trade-off of this approach, beyond limited depth of testing, is that it may be costly to keep up to date with the entity being mocked up (AWS here). The details depend on the pace of evolution of the original and the desired scope of testing (such as if we consider updated external constraints as important or not, example being AWS commissioning and decommissioning locations and instance (VM) types).

An example sequence diagram for CSEM is given in Figure 2 – Example CSEM Sequence Diagram. It depicts a typical usage scenario where an end user utilises IaC tooling (such as the previously-mentioned Terraform or Ansible) configured to use CSEM and then inspects the results from the perspective of the tooling as well as CSEM.

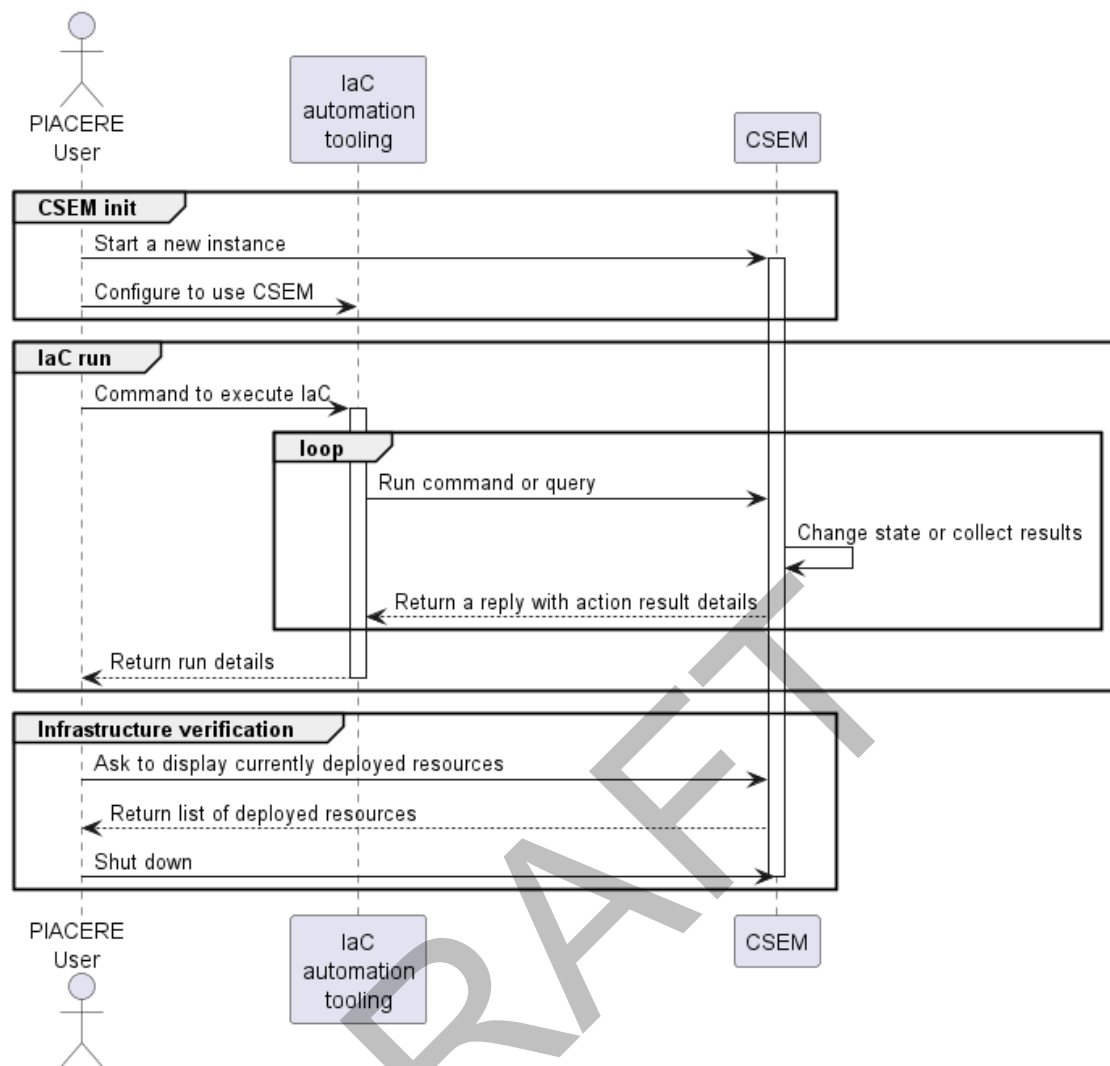


Figure 2 – Example CSEM Sequence Diagram

2.3 Main innovations

The main innovation of the CSEP is the opinionated deployment of OpenStack that is accessible via an easy-to-use REST API, integrated also with a common secrets' store in the form of Hashicorp Vault².

Regarding CSEM, the main innovation is the enhanced AWS mocking service allowing to simulate and validate IaC runs without actually querying the paid service.

² <https://www.vaultproject.io/>

3 Overview of preliminary experiments

The final versions of CSE tools have yet to be applied in the use cases. These were tested experimentally mostly internally and the following stems from our experience with them.

Firstly, on the provisioner, it has been used successfully to create ad-hoc, testing deployments of OpenStack by people new to OpenStack. The deployments spanned from single to a few (~5) machines, both physical and virtual as well as a mix of the two. The plug-and-play (or more like run-and-play) approach and the easy-to-use API have been proven successful in enabling more users to create OpenStack environments to interact with by themselves in a sandbox way. That said, during the experiments, we have encountered issues due to the evolving nature of OpenStack and supported Linux distributions which required us to update the dependencies (in this case, Kolla Ansible) and apply relevant patches to these dependencies to keep the core provisioner logic unaffected. Thankfully, these tasks were simple to carry out in the proposed solution and did not impact the testing considerably.

Secondly, on the Mocklord, it has been used successfully to simulate the runs of example IaC provided early in the project by the Posidonia use case which uses AWS. In fact, its development was driven mostly by that example. Additionally, Mocklord has been used for internal testing of AWS IaC at 7bulls.com. These examples were more complex than the initial tested example and they have exposed certain (expected) shortcomings in this simulated approach to IaC testing which are reported in the next section.

DRAFT

4 Lessons learnt and outlook to the future

Regarding the provisioner, what we have observed and what has been described in the previous section, is that it is useful as a tool which hides away complexities of OpenStack at the cost of much lower flexibility. This gives users the confidence that the basics will just work out of the box and that's enough for non-production environments. The tool depends highly on the OpenStack ecosystem (including supported Linux distributions) and the used tooling (Kolla Ansible) which are both evolving parallelly, so it is important to stay in sync with them to keep the necessary level of compatibility going forward. Again, thankfully, the proposed solution has thus far proven to be simple to keep up-to-date and no show-stoppers are expected. On the other hand, no other deployment target than OpenStack has emerged as a popular-enough solution to warrant its implementation in the provisioner which limits its current scope to this single "provider".

Regarding the Mocklord, or the general simulated approach to IaC testing, it has proven difficult to scale to much more complex scenarios where more specific AWS resources were applied with complex interdependencies. Its success depends strictly on ongoing support and development of the validation rules relevant to the tested IaC so that it is properly covered – to eliminate not only false positives but also false negatives (i.e., when the tool rejects an otherwise-valid IaC). The only public effort in this scope involves the AWS APIs (as covered here) and this makes a general solution even more problematic because other providers are not covered even in very basic scenarios. Considering the support cost of a single provider and the fact that this mostly benefits this single provider and its users, it seems unlikely that this particular approach would be continued without direct effort from the interested parties (most notably the cloud providers who hold the "ground truth"). When this tool's goal is to cut the costs related to cloud providers, it's easy to say it is not in their interest to collaborate and thus this solution might look like it does not have a bright future. However, the upstream project³ is pretty active and merges contributions from multiple independent collaborators which gives hope that this kind of testing can be achieved collaboratively at least for this one cloud provider.

³ moto, <https://github.com/getmoto/moto>, discussed further in the appendix.

5 Conclusions

This deliverable has described the PIACERE approach to Canary Sandbox Environment, which is to provide both real (non-simulated) and simulated Canary Resource Providers for dynamic testing of IaC in a sandbox-like environment.

The currently provided prototypes offer both the provisioning functionality via Canary Sandbox Environment Provisioner (CSEP), and the mocked approach to testing AWS-compatible IaC via the Canary Sandbox Environment Mocklord (CSEM) which avoids incurring costs and providing any extra infrastructure.

All relevant information has been provided in this single document, giving also an outlook to the future and discussing preliminary results and lessons learnt. The provisioner's usability is high and its support depends on keeping in sync with the upstream OpenStack community. On the other hand, the Mocklord is more problematic because of a relatively high support cost to effect ratio and the related limited scope and applicability.

DRAFT

6 References

- [1] Hashicorp, 'Terraform by HashiCorp'. <https://www.terraform.io/> (accessed Mar. 29, 2021).
- [2] 'ansible/ansible'. Ansible, Feb. 18, 2022. Accessed: Feb. 18, 2022. [Online]. Available: <https://github.com/ansible/ansible>
- [3] 'etcd'. etcd-io, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: <https://github.com/etcd-io/etcd>
- [4] S. Ramírez, 'tiangolo/fastapi'. Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: <https://github.com/tiangolo/fastapi>
- [5] 'encode/uvicorn'. Encode, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: <https://github.com/encode/uvicorn>
- [6] 'The OpenAPI Specification'. OpenAPI Initiative, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: <https://github.com/OAI/OpenAPI-Specification>
- [7] 'swagger-api/swagger-ui'. Swagger, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: <https://github.com/swagger-api/swagger-ui>
- [8] 'redoc'. Redocly, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: <https://github.com/Redocly/redoc>
- [9] S. Colvin, 'pydantic'. Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: <https://github.com/samuelcolvin/pydantic>
- [10] 'hvac'. hvac, Nov. 12, 2022. Accessed: Nov. 14, 2022. [Online]. Available: <https://github.com/hvac/hvac>
- [11] 'openstack/kolla-ansible'. OpenStack, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: <https://opendev.org/openstack/kolla-ansible>
- [12] S. Pulec, 'spulec/moto'. Mar. 27, 2021. Accessed: Mar. 21, 2021. [Online]. Available: <https://github.com/spulec/moto>
- [13] 'Boto3 - The AWS SDK for Python'. the boto project, Nov. 13, 2022. Accessed: Nov. 14, 2022. [Online]. Available: <https://github.com/boto/boto3>
- [14] 'PyCQA/bandit'. Python Code Quality Authority, Nov. 13, 2022. Accessed: Nov. 15, 2022. [Online]. Available: <https://github.com/PyCQA/bandit>
- [15] 'GitLab CI/CD | GitLab'. <https://docs.gitlab.com/ee/ci/> (accessed Nov. 15, 2022).
- [16] 'Mypy: Static Typing for Python'. Python, Nov. 15, 2022. Accessed: Nov. 15, 2022. [Online]. Available: <https://github.com/python/mypy>
- [17] 'PDM'. PDM, Nov. 15, 2022. Accessed: Nov. 15, 2022. [Online]. Available: <https://github.com/pdm-project/pdm>
- [18] 'tox automation project'. tox development team, Nov. 15, 2022. Accessed: Nov. 15, 2022. [Online]. Available: <https://github.com/tox-dev/tox>
- [19] 'Gherkin'. Cucumber, Nov. 15, 2022. Accessed: Nov. 15, 2022. [Online]. Available: <https://github.com/cucumber/gherkin>
- [20] 'The Moby Project (Docker)'. Moby, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: <https://github.com/moby/moby>
- [21] 'Docker Compose'. Docker, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: <https://github.com/docker/compose>
- [22] 'Kubernetes API Reference', *Kubernetes*. <https://kubernetes.io/docs/reference/> (accessed Nov. 09, 2021).

APPENDIX: Implementation, delivery and usage

1 Implementation

1.1 Fitting into overall PIACERE Architecture

CSE tools live in the runtime phase of the PIACERE framework. They have been depicted at the bottom of Figure 3 - PIACERE Runtime Diagram for Deployment (from D2.2) – a diagram provided by Work Package 2.

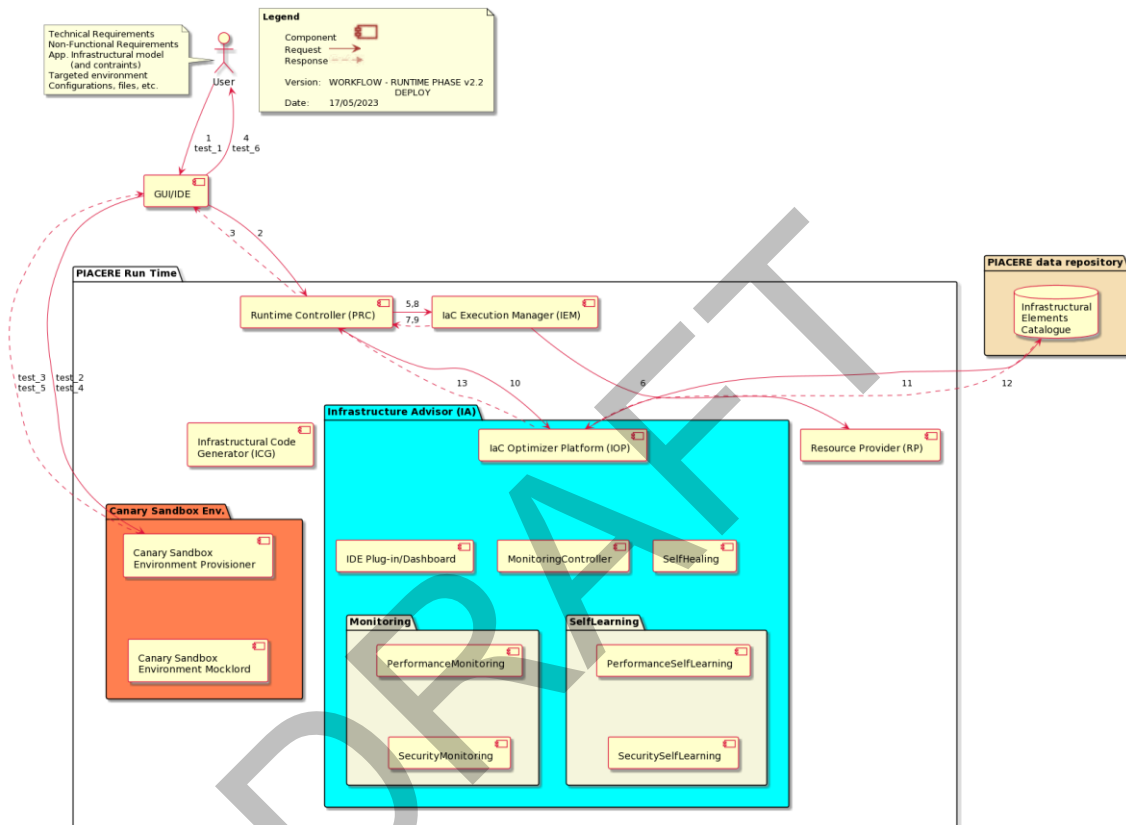


Figure 3 - PIACERE Runtime Diagram for Deployment (from D2.2)

CSEP is to be used by the user, most likely via the IDE user interface, to create Canary Resource Providers. The Canary Resource Providers are to be used by the IEM (IaC Execution Manager) to execute IaC against. The CSEP is a helper component that exposes the interface to do the provisioning and the CSEM is our implementation of mocked-up resources provider (a special type of Canary Resource Provider).

1.2 Technical description

1.2.1 Canary Sandbox Environment Provisioner - CSEP

1.2.1.1 Prototype architecture

The prototype architecture is shown in Figure 4 - CSEP Architecture Diagram. The following components make up the CSEP prototype:

- CSEP API
- CSEP worker and its backends
 - dummy CSEP worker backend
 - OpenStack CSEP worker backend

Additionally, in the proposed architecture, a shared database is used for storing exchanged data. Optionally, Hashicorp Vault can be used to enable secure credentials storage for use by CSEP worker.

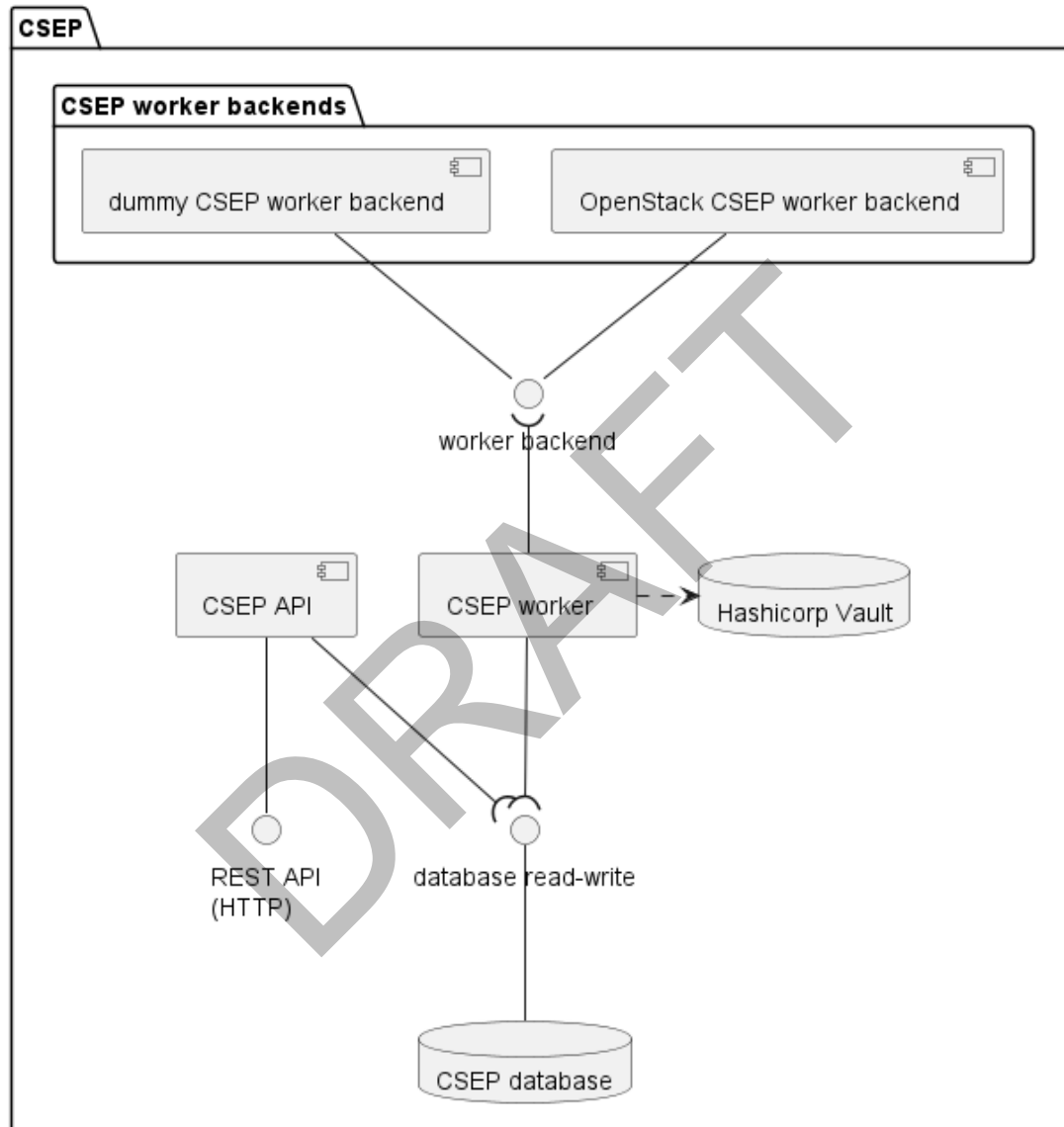


Figure 4 - CSEP Architecture Diagram

The proposed architecture allows for extensibility via the backends system of the worker – new deployment types can be added at will and can reuse the existing, common code.

1.2.1.2 Components description

The CSEP **API component** is the integration point with the external world and any other component from the PIACERE framework, currently imagined to be the IDE. This component's role is to expose the REST API (via HTTP) offered by CSEP to let CSEP users provision CRPs using CSEP. The REST API accepts and outputs JSON objects.

The CSEP **worker component** is responsible for the actual provisioning of the desired type of deployment. It calls and manages the necessary tooling to achieve this goal. Two backends are provided: dummy (for integration testing purposes) and OpenStack.

1.2.1.3 Technical specifications

As is common in the PIACERE project's software, CSEP is provided as a containerised daemon which solves the issue of installation for the end user. However, the technical details are nonetheless important for the development of it and thus are described in detail in this section.

The prototype is written in the Python language, for the version 3.10, against which it is tested both statically and dynamically.

Both provided components utilise the etcd [3] database, currently in version 3.5, and thus also the accompanying library (named etcd3) to connect to it. The etcd database is a simple, strongly consistent and distributed key-value store, the project itself is under the Cloud Native Computing Foundation (CNCF) umbrella.

The API component utilises the FastAPI [4] framework. This framework supports Asynchronous Server Gateway Interface (ASGI) which must be understood by the web server in use. The used web server is the most often recommended one nowadays – uvicorn [5]. FastAPI relies heavily on Python's type annotations to drive API definition and its validation. FastAPI also offers native generation of OpenAPI specification [6] and is capable of rendering docs in Swagger UI [7] as well as Redoc [8]. Of indirect dependencies, a notable one is Pydantic [9]. Pydantic offers the framework for extra type annotations and the actual validation in instantiated objects.

The worker component reuses the Pydantic library used in the API component for easy serialisation-deserialisation from the common database. For its own operation, it also uses the hvac [10] library. This library provides an easy and popular way of access to the Hashicorp Vault's API which is an optional component to enable a secure access to credentials.

The worker component's backend is based on the deployment solutions relevant to the available options. For this revision, OpenStack is supported to be deployed with the help of Kolla Ansible [11]. Kolla Ansible is a community-powered OpenStack deployment project which utilises Docker containers to deploy production-ready OpenStack, thus it aligns nicely with the containerised approach of deployment in PIACERE.

Both components, as well as the shared database, are containerised and thus the details of the underlying platform are less relevant. Any sufficiently modern kernel which is able to run Docker images will suffice.

1.2.2 Canary Sandbox Environment Mocklord – CSEM

1.2.2.1 Prototype architecture

The architecture of CSEM is based on the moto [12] library. The moto library has been included in the SOTA in version 1 of this deliverable (see the appendix). Since then, the landscape has not changed, and we have concluded that the amount of effort necessary for the implementation of mocking a cloud provider is beyond the scope of this project. Moreover, the use case partners of the PIACERE project are all either based on Amazon Web Services (AWS) or an on-premises solution. Thus, the approach has focused entirely on AWS and moto. To this end, we have enhanced this open-source library and sent our changes upstream so that others can benefit directly from our effort. More so, it means that the enhancements we have proposed will be kept up by other teams of developers, possibly improving the solution even further in the desired direction. Since CSEM is essentially a single component, the architecture diagram has been omitted as it would present no tangible purpose.

1.2.2.2 Components description

As mentioned in the previous subsection, CSEM consists of a single component – CSEM itself – which packages the moto library with necessary configuration. The details on it are presented in the next subsection.

1.2.2.3 Technical specifications

As mentioned already, CSEM is based on the moto library. The moto library was born for the need of automated testing of the boto3 [13] library which allows easy access to AWS APIs from Python. Of utmost importance to CSEM is what moto calls the server mode. This functionality allows to start an HTTP service that mocks (acts like) the AWS APIs and is available for external HTTP requests, as opposed to being available for the Python testing infrastructure. Server mode tracks the current status of the mocked platform which makes it possible to be interacted with by tooling which assumes persistence, such as Terraform.

During testing of the usefulness of this approach, we were basing our actions on an example provided by one of the use case partners. This has revealed a series of issues (bugs or missing features that PIACERE users would expect) and so we have developed (and made available upstream) code to mitigate them. The following subsections group and discuss the various changes made to moto, along with the reasoning. Most activities required inspecting the actual behaviour of the AWS APIs.

1.2.2.3.1 Validations

As moto's original aim is to be used as a more rigorous unit testing facility for libraries like boto3, it has already included validation capabilities to some extent. However, they were mostly targeted at what we would call grammar correctness – that the requests are well-formed and not missing required elements. As such, actual validation of the values was of lesser priority. However, CSEM is meant to verify correctness of IaC before deployment to target environment. Thus, validating the values of parameters, that would be later validated by AWS, becomes important. To this end, we have implemented extra validations in terms of availability zones matching regions, instance types matching zones, keypair existence and AMI (Amazon Machine Image) existence. In case of wrong values, the mocked API returns the same error code that AWS does. The following paragraphs discuss the validations in more detail.

Availability zone check ensures that availability zone that is assigned to requested instances is valid for the region selected for the instance, for example: at the moment, us-west-1 region has availability zone us-west-1a but not us-east-1a nor us-west-1d.

Instance type validation validates that provided instance type name points to an existing instance type and that the selected instance type is available for assigned region and availability zone.

Keypair is a resource created by the cloud user while AMI may be an existing one or created by the user. In any case, the requested instance must use an existing keypair and AMI to be accepted. This requirement has been added in moto as well.

Validating features in moto are disabled by default to avoid breaking any pre-existing applications utilizing moto. They can be enabled by setting associated environment variables. These are set automatically as enabled by CSEM.

1.2.2.3.2 Record and replay

Validations are crucial for the operation of CSEM and some of them, such as keypair and AMI validations, rely on the state. Other existing validations, that e.g. operate on network resources, also rely heavily on current state. However, the current state of the moto server was not preservable and thus only from-scratch scenarios could be reliably tested – scenarios such as redeployments and scaling were not easily achievable. In general, we could say that IaC moves the state of some deployment from A to B. In moto server, the A would have to be the null state or achieved by a previous run of another IaC.

To alleviate this issue, we proposed the record-and-replay feature. Requests sent to the server may change its state. Future requests can result in different outcomes depending on the state of the server established by previous requests. The introduced recorder stores relevant information about each request sent to moto. It offers a possibility to download a file with the recording or upload a recording generated in the past, on a different machine or even created manually. Then the feature allows to replay all the recorded requests in the same order, with the same parameters. All functionalities are exposed by the moto server API. The recorder is enabled by default in CSEM.

The end goal of this functionality could have also been implemented using serialisation of the internal structure. This would normally be more reliable. However, for one we don't know the exact implementation of the internal structures of AWS. Also, the current implementation in moto is not easily serializable nor guaranteed to remain stable and thus the alternative of record-and-replay was chosen over it.

1.2.2.3.3 Seeding the RNG

The recording and replay allow to recreate the state. However, both the real and mocked AWS APIs rely on (universally) unique identifiers which are generated using a random number generator (RNG). This results in replay sessions not being 1:1 with the previous outcome w.r.t to these identifiers. To mitigate this, we have added the capability to set the seed affecting all identifier-generating functions.

1.2.2.3.4 Compatibility fixes

We had to implement fixes to ensure that the use case's example, that succeeds on AWS, would succeed on moto too. The following paragraphs describe the various fixed areas that enhance the overall correctness of moto.

AWS uses Amazon Resource Names (ARNs) to uniquely identify AWS resources. ARN is required when there is a need to specify a resource unambiguously across all of AWS and may be (and more often than not is) used by IaC tooling such as Terraform. A bug in moto caused ARN to be missing from the response for DBSubnetGroup. Despite the fix being a simple one, the actual debugging investigation was costly as moto's replies are not 1:1 with AWS (and cannot easily be

1:1) and we were relying on IaC run via third-party software (Terraform): the incomplete response was initially accepted but caused a cascade of wrong requests being sent afterwards.

Networking config is a crucial part of complex IaC. Route tables is what drives the core of networking in case of AWS. Despite this, there were several bugs in the handling of route table associations which is a way to connect route tables to the resources that use them (subnets): main association was returned for route tables without main association, it was not possible to replace the main association and the API response to request for replacement was missing association state. These bugs also caused the example use case IaC to fail in moto.

1.2.2.3.5 Prettifying responses and the quest for 1:1 behaviour with AWS

Finally, in the process of implementing the other features and fixes, we noticed one particular difference between the AWS and moto – the formatting of results. It is worth noting that the AWS API is RESTful and uses XML as the data interchange format. In principle, any information in XML could be written in a single text line and it was the case with moto. Nonetheless, AWS returns the output “prettified” which means it is standardised for human readability: with new lines for XML elements, proper spacing and indentation. Thus, to this end, the ability to “prettify” the response was added to moto. The option is disabled by default to avoid breaking pre-existing applications of moto and incurring the cost of prettifying (as the content has to be parsed).

An important factor is the 1:1 compatibility with AWS. While the actual behaviour cannot be followed 1:1 as internals are unknown, the principle that we and others before us adopted is to follow the black box comparison approach – the same inputs are used against both AWS and moto, and the replies from the two are compared to then ensure that moto behaves more like AWS. This works but is limited to the example being put under scrutiny. Thus, most generalisation attempts are costly.

2 Delivery and usage

2.1 Canary Sandbox Environment Provisioner - CSEP

2.1.1 Package information

Package's root directory contents include 7 directories and 17 files. The files are:

- .bandit – a file to configure one of the code linters (bandit [14]),
- .dockerignore – a file used by Docker to ignore certain paths from being included in the build process (e.g., to avoid copying temporary, helper and development files),
- .gitignore – a file relevant to Git (repository) operations, containing file paths to ignore,
- .gitlab-ci.yml – a file configuring the GitLab CI/CD [15] solution used throughout the PIACERE project for its continuous development,
- Dockerfile – a file containing the recipe to build the container image which supports the software,
- LICENSE – the license file (MPL 2.0),
- README.md – a quick documentation with instructions on how to install, run and use,
- docker-compose.override.yml – a default override file for docker-compose to obtain a development- and testing-friendly default run while providing a base for extension,
- docker-compose.yml – a YAML file used by docker-compose to quickly deploy the software to Docker (its base enhanced by the override mentioned above),
- mypy-requirements.txt – Python (pip) requirements file for the mypy [16] linter which validates static typing,
- openapi.json – a JSON file with OpenAPI specification dump from the current version of software,
- pdm.lock – a file used by PDM (Python Dependency Management) [17] tooling for ensuring that dependencies are locked; in CSEP, PDM has replaced Pipenv which we used initially in v1,
- pyproject.toml – the main Python project configuration file holding configuration of the linters and declaring dependencies,
- requirements.txt – the rendering of project's dependencies from pyproject.toml in a format consumable by pip,
- test-requirements.txt – additional requirements for unit tests,
- tox.ini – the configuration for the tox [18] tool which allows for easy creation of test and development Python environments,
- uvicorn-requirements.txt – additional requirements for the default and preferred server running the API service.

and the directories are:

- KR8-features – holds the definition of features implemented in the project, described in the Gherkin [19] language,
- csep_api – holds the main module for the API component,
- csep_common – holds the module shared between both components,
- csep_worker – holds the main module for the worker component,
- openstack_requirements – holds files necessary for proper and repeatable builds of the used Kolla Ansible tooling,
- patches – holds diff patches applied to external dependencies (currently Kolla Ansible) that have not yet been upstreamed; this has been added to have more independence from upstream and progress quicker,
- tests – hold files relevant to testing, including unit tests and examples,

- tools – miscellaneous tools, mostly useful during development.

2.1.2 Installation instructions

The software is containerisable and the recipe is provided along with an example docker-compose deployment. The users are recommended to install relatively modern Docker [20] (19.03+) and docker-compose [21] (1.29+). Then it is only a matter of running

```
docker-compose up -d
```

to get the software running. Please note that the image might take a while to build before it can be used. The docker-compose command above will start 4 containers corresponding to the two components, the database and the Hashicorp Vault (optional, can be disabled if not desired).

2.1.3 User Manual

If the installation instructions were followed, the CSEP API will be exposed at 127.0.0.1:8000. Navigating to <http://127.0.0.1:8000/docs> will show Swagger UI with the current OpenAPI specification rendered as seen in Figure 5 - API endpoints as shown by Swagger UI at /docs. The requests can be issued directly from there, but we recommend the use of Postman⁴ for more flexibility. There are 9 endpoints in total – 5 for queries (i.e., getting information from the CSEP):

- GET /deployments/ – this will retrieve information on all the deployments,
- GET /deployments/{deployment_name} – this will retrieve information on the chosen deployment (as indicated by the {deployment_name} path variable),
- GET /deployments/{deployment_name}/events – this will retrieve the events that apply to the chosen deployment,
- GET /deployments/{deployment_name}/events/last_message – this will retrieve the message of the last event that applies to the chosen deployment; it usually carries the necessary details on why the deployment failed,
- GET /deployments/{deployment_name}/custom_output/{custom_output_type} – this will retrieve custom output from the backend; it is backend-specific; currently, it supports `clouds.yaml` as the custom output type for the OpenStack backend which produces output in this format which includes connection and authentication details necessary for OpenStack clients, including Terraform,

and 4 for commands:

- POST /deployments/ – this will issue a new deployment request to CSEP so that the CSEP worker should ensure it will be acted upon,
- DELETE /deployments/ – this will remove the information about the chosen deployment (as indicated by the {deployment_name} path variable) from the CSEP database; it does not do “undeployment” or “cleanup” which are covered by the *undeploy* endpoint described below,
- POST /deployments/{deployment_name}/redeploy – this will issue a redeployment request to CSEP so that the CSEP worker should ensure it will be acted upon; it is similar to initial deployment but allows the user to reuse all the settings and only patch details as necessary,
- POST /deployments/{deployment_name}/undeploy – this will issue an undeployment request to CSEP so that the CSEP worker should ensure it will be acted

⁴ <https://www.postman.com/>

upon; the worker will undeploy, i.e., undo, the deployment and clean up relevant resources so that they can be reused.

GET	/deployments/	Get All Deployments	▼
POST	/deployments/	Post Deployment	▼
GET	/deployments/{deployment_name}	Get Deployment	▼
DELETE	/deployments/{deployment_name}	Delete Deployment	▼
GET	/deployments/{deployment_name}/events	Get Deployment Events	▼
GET	/deployments/{deployment_name}/events/last_message	Get Deployment Events Last Message	▼
GET	/deployments/{deployment_name}/custom_output/{custom_output_type}	Get Deployment Custom Output	▼
POST	/deployments/{deployment_name}/redploy	Redeploy Deployment	▼
POST	/deployments/{deployment_name}/undeploy	Undeploy Deployment	▼

Figure 5 - API endpoints as shown by Swagger UI at /docs

Example request bodies are present in `tests/sample_requests` in the main directory (see Figure 6 - an example API request (POST new dummy deployment)). The fields in the API should be self-explanatory. The CSEP API was loosely modelled after Kubernetes API [22] and objects provide two major keys: `spec` and `status`. The `spec` defines the desired state and `status` shows the current state. The `status` cannot be set by the API user and thus is not present in the requests, only responses. CSEP supports multiple types of deployments and thus the deployment type is provided in the `spec` (in `type` field). Currently, only two values are supported:

- `dummy` – for testing purposes (note lowercase “d”),
- `OpenStack` – for deploying OpenStack.

Types may change the parameters available in the `spec`. There are, however, certain fields which are always present:

- `auth` – defines the way to authenticate when connecting to the target hosts; actual possible methods depend on deployment type,
- `hosts` – defines the target hosts and how to reach them (plus extra metadata that might be applicable to certain deployment types).

More details are available from the aforementioned API documentation available at `/docs` path from a deployed API instance.

```
{
  "name": "test",
  "spec": {
    "type": "dummy",
    "auth": {
      "type": "username_password",
      "username": "test",
      "password": "xyz"
    },
    "hosts": [
      {
        "ip_address": "127.0.0.1"
      }
    ]
  }
}
```

Figure 6 - an example API request (POST new dummy deployment)

As already mentioned, an API response will include also the status of the deployment as seen in Figure 7 - an example API response (POST new dummy deployment). The most important field in that status is the progress which summarises the current state of the deployment. The status also tracks events that happen to the deployment (such as the different stages).

```
{
  "name": "test",
  "spec": {
    "auth": {
      "type": "username_password",
      "username": "test",
      "password": "xyz"
    },
    "hosts": [
      {
        "ip_address": "127.0.0.1"
      }
    ],
    "type": "dummy"
  },
  "uuid": "bac855d4-acb3-4d05-953a-98d8d85f7fb0",
  "created": "2021-11-15T11:57:04.919988",
  "status": {
    "last_updated": "2021-11-15T11:57:04.919988",
    "progress": "New"
  }
}
```

Figure 7 - an example API response (POST new dummy deployment)

2.1.4 Licensing information

The license of CSEP is the **Mozilla Public License, version 2.0 (MPL 2.0)**. The licenses of dependencies and compatible third-party components are compatible with MPL 2.0, mostly comprising BSD, MIT, Apache 2.0 and MPL 2.0 licenses. For the major ones, these are the licenses (sorted alphabetically):

- Ansible – GPL 3.0
- etcd3 (python library) – Apache 2.0
- etcd3 (the database service) – Apache 2.0
- FastAPI – MIT
- hvac – Apache 2.0
- Kolla Ansible – Apache 2.0
- Pydantic – MIT
- PyYAML – MIT
- Unicorn – 3-clause BSD
- Vault – MPL 2.0

The license is included in the CSEP git repository, in the LICENSE file.

2.1.5 Download

The most recent public code is generally available at Tecnia's GitLab, in the public part of the PIACERE project: <https://git.code.tecnia.com/piacere/public/the-platform/cse/csep>

2.2 Canary Sandbox Environment Mocklord - CSEM

2.2.1 Package information

Package's root directory contents include 1 directory and 7 files. The files are as following:

- .dockerignore – a file used by Docker to ignore certain paths from being included in the build process (e.g., to avoid copying temporary, helper and development files),
- .gitignore – a file relevant to Git (repository) operations, containing file paths to ignore,
- .gitlab-ci.yml – a file configuring the GitLab CI/CD [15] solution used throughout the PIACERE project for its continuous development,
- Dockerfile – a file containing the recipe to build the container image which supports the software,
- LICENSE – the license file (MPL 2.0),
- README.md – a quick documentation with instructions on how to install, run and use,
- docker-compose.yml – a YAML file used by docker-compose to quickly deploy the software to Docker,

and the only directory is “examples” which contains the Terraform examples that will run against CSEM. Both working and failing examples are included.

2.2.2 Installation instructions

The software is containerisable and the recipe is provided along with an example docker-compose deployment. The users are recommended to install relatively modern Docker [20] (19.03+) and docker-compose [21] (1.29+). Then it is only a matter of running

```
docker-compose up -d
```

to get the software running. Please note that the image might take a while to build before it can be used. The docker-compose command above will start 1 container serving CSEM (the moto server patched and configured to enable CSEM operations).

2.2.3 User Manual

After the software is installed and started (see the previous subsection for instructions), it can be, by default, reached at <http://127.0.0.1:8000/> where it offers the API typical of AWS. It is possible to use this API with any AWS-compatible client, and in PIACERE we choose Terraform to showcase its capabilities as Terraform is one of the supported IaC solutions in the PIACERE project.

To use CSEM in Terraform, one has to configure the AWS provider overriding the real AWS endpoints and disabling certain checks which do not make sense in the context of CSEM. Additionally, setting access credentials is irrelevant (and obviously the production ones should not be used as they should be protected at all costs). Thus, in case of testing the EC2 capabilities only, it is possible to add this stanza to the Terraform definitions:

```
provider "aws" {  
  # moto does not care  
  access_key = "whatever_access_key"  
  secret_key = "whatever_secret_key"  
  
  # the default region, not too relevant in here  
  region = "us-east-1"  
  
  # do not make sense with moto  
  skip_metadata_api_check = true  
  skip_credentials_validation = true  
  skip_requesting_account_id = true  
  
  endpoints {  
    ec2 = "http://127.0.0.1:8000"  
  }  
}
```

The most important is the “endpoints” stanza which will ensure that Terraform does not try to contact the AWS endpoints and will use CSEM instead.

Following that, the user can include the desired resource declarations, such as the ones given in the examples present in the package. The expected behavior of Terraform should ensue after running the typical plan-apply cycle.

Apart from the AWS API, the tool allows to browse the contents using the <http://127.0.0.1:8000/moto-api> path. In there, the user can see what has been created internally in CSEM. Moreover, since the recording feature has been enabled by default on start, all the (at least potentially) state-changing requests are recorded to a temporary file. The file can be accessed and manipulated directly at /tmp/moto.json in the container or using the procedures described in the contributed moto help⁵. For reproducible runs, we recommend

⁵ <http://docs.getmoto.org/en/latest/docs/configuration/recorder/index.html>

starting each CSEM session with a POST request to <http://127.0.0.1:5000/moto-api/seed?a=123> where 123 is the desired seed (its actual value is irrelevant as long as it stays the same).

2.2.4 Licensing information

The license of CSEM is the **Mozilla Public License, version 2.0 (MPL 2.0)**. The license of the main, dependency, moto, is Apache 2.0 which is compatible with MPL 2.0.

The license is included in the CSEM git repository, in the LICENSE file.

2.2.5 Download

The most recent public code is generally available at Tecnalia's GitLab, in the public part of the PIACERE project: <https://git.code.tecnalia.com/piacere/public/the-platform/cse/csem>.

DRAFT