PIACERE

Deliverable D5.5

Canary Sandbox Environment prototype – v2

Editor(s):	Radosław Piliszek (7bulls)
Responsible Partner:	7bulls
Status-Version:	Final-v1.0
Date:	25.11.2022
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	30.11.2022

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

Abstract:	The new main outcomes of Task 5.2 - PIACERE Canary environment for IaC behaviour testing and simulation - are presented in this deliverable. This deliverable corresponds to Key Result 8 – Canary Sandbox Environment. This is v2 of the deliverable D5.4 and presents the progress made on top of D5.4 in the second year of the project.
Keyword List:	Canary environment, sandbox, dynamic testing
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	12.10.2022	First draft version.	Radosław Piliszek (7bulls)
v0.2	16.11.2022	Extra content regarding changes in v2.	Radosław Piliszek, Marcin Bartmański (7bulls)
v0.3	24.11.2022	Fixes and enhancements after the internal review.	Radosław Piliszek (7bulls)
v1.0	25.11.2022	Ready for submission	Juncal Alonso (TECNALIA)

Table of contents

List of tables

List of figures

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)
SOTA	State of the Art
TSR	Technical Specification Report (e.g., this document)

Executive Summary

This document is the Technical Specification Report (TSR) of the second 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.

As this (D5.5) is version 2 of "Canary Sandbox Environment prototype" TSR, it's based on version 1 of the same-named deliverable D5.4. The content has been enhanced, incorporating further work that has happened in T5.2, focusing mostly on the Mocklord (CSEM) component, but also introducing various improvements to the previously proposed tool – the Provisioner (CSEP). The version 1 document's structure has been mostly preserved to ease with the interpretability of the work results. However, the section on State of the Art (SOTA) has been moved to the annexes as there are no new findings to report but we strived to make this document self-contained.

The document starts with an introduction and then moves onto discussing the internals and interfaces of the delivered tools. Each main section is divided into subsections for the Provisioner and the Mocklord, respectively. The section on implementation describes the architecture of the proposed tooling and its realisation. The section on delivery and usage describes how the tooling can be obtained and used. Finally, the conclusions complete the TSR by summarising the outcomes and discussing future work.

The tooling reported in this deliverable, in this second revision, still enables users to provision OpenStack which can then be utilised by the IaC Execution Manager (IEM). Additionally, in v2, PIACERE users are 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. The tools' motivation, architecture and technical aspects are all discussed in detail.

The future final version of the Canary Sandbox Environment tooling will be further adapted to project and projects' use cases' needs. The D5.6 deliverable (v3, the final one; due month 30) will describe it.

1 Introduction

1.1 About this deliverable

This document is the Technical Specification Report (TSR) of the second 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 second 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.

1.2 Document structure

The TSR is organised in 5 main sections which are further organised in subsections. The 1st section is this introduction containing the information on the TSR itself. The 2nd section describes the architecture of the proposed solution. The 3rd section details the steps to obtain and consume the software. The 4th section concludes the document. The document ends with an appendix with SOTA from v1 of this deliverable and a list of references.

Sections 2 and 3 include also a summary of changes that happened since the v1 of the deliverable.

2 Implementation

2.1 Changes in v2

The version 2 of CSE introduces the Mocklord (CSEM). CSEM was already advertised in v1 of this deliverable, but its implementation details remained unknown at time. We have carried out research and experimentation in terms of viability of its implementation and have settled on an implementation based on the moto library presented before (in v1) in the SOTA. The implementation CSEM is described in detail in subsections 2.2.2 and 2.3.2.

The changes in this section reflect progress on the implementation of CSEM as well as updates to CSEP to increase its security, robustness and usability properties, after the initial validation by the use cases and other internal testing both in the consortium and the KR's owner company (7bulls.com). In that regard, the major points are that the CSEP:

- has been enhanced with an integration with Hashicorp Vault [1] to allow for secure credentials storage and access,
- had its API adapted to make it more user-friendly – certain details have been moved to other API paths,
- has gained the ability to redeploy, i.e., run the same, or slightly modified, deployment action without the need to define the entire deployment again.

Other than that, the high-level design and operation of CSEP has not changed considerably and all the details are preserved for completeness. Beyond updates to the functionality, there are also updates regarding the dependencies which have been captured in this deliverable under "technical specifications" and "package information".

Finally, the Requirements table has been updated to reflect the current status of the software being delivered.

2.2 Functional description

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. However, the CRPs are general enough and can be used by any compatible tooling directly (such as Terraform [2] or Ansible [3]).

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 efforts[1], 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. The requirements

[1] Reported in D2.1 and D2.2.

themselves have not changed since v1 of the deliverable but the status of some of them has. The table comments on the updates made for v2.

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

Req ID	Description	Status at M24	Comment at M24
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	This is extended to CSEM in v2.
REQ34 / WP5.2-REQ2	CSE to keep track of and allow querying of the deployment state to allow comparison against the expected one.	Implemented	This is extended to CSEM in v2.
REQ37 / WP5.2-REQ3	CSE to have a simulated mode limited to provisioning.	Implemented	This is extended to CSEM in v2.
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	(already in v1)
REQ39 / WP5.2-REQ5	CSE to enable extensibility (documented way): adding new mocked services, adding new "real" deployments.	Implemented	This is extended to CSEM in v2.

The main innovation of the CSEP is the opinionated deployment of OpenStack that is accessible via an easy-to-use REST API. In this cycle, CSEP has been enhanced with a secondary innovation of integration with a common secrets' store in the form of Hashicorp Vault. This is to improve the multi-user security of the PIACERE framework as a whole as requested in non-functional requirements REQ10 ("The communication within the different components of the architecture should be done in a secure way (e.g. https, keycloak).") and REQ88 ("PIACERE framework should be usable by a team of people collaborating in the development of the same IaC.") targeted at the KR13 (the PIACERE framework), spanning the entire PIACERE ecosystem of tools.

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

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.

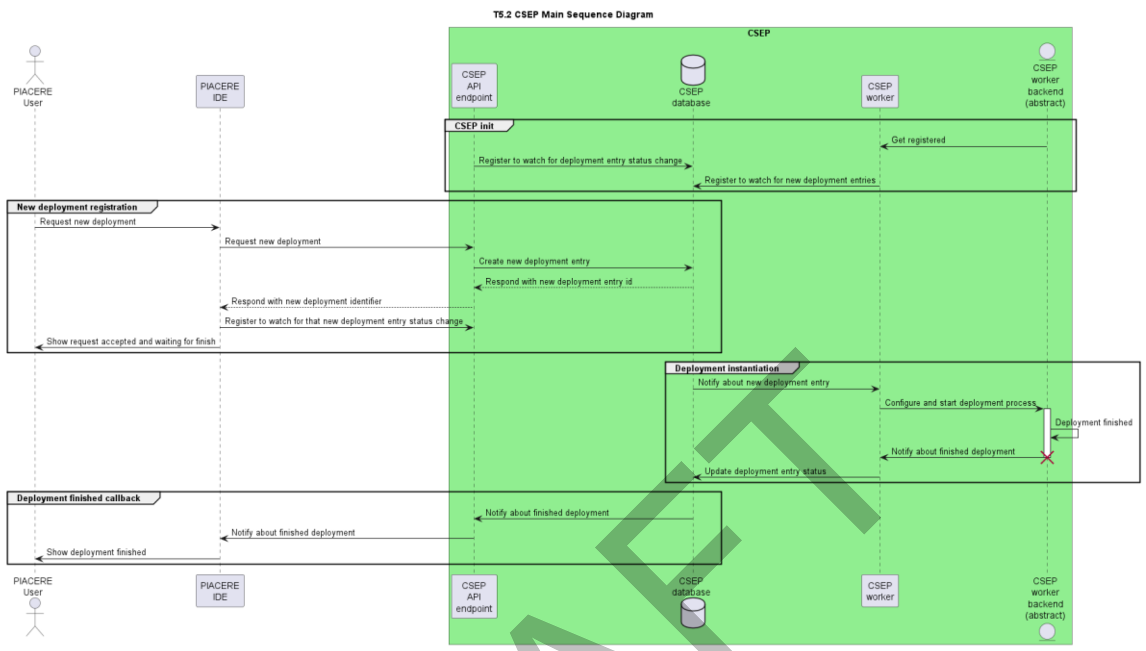
T5.2 CSEP Main Sequence Diagram
CSEP
PIACERE User
PIACERE IDE
CSEP API endpoint
CSEP database
CSEP worker
CSEP worker backend (abstract)
CSEP init
Get registered
Register to watch for deployment entry status change
Register to watch for new deployment entries
New deployment registration
Request new deployment
Request new deployment
Create new deployment entry
Respond with new deployment entry id
Respond with new deployment identifier
Register to watch for that new deployment entry status change
Show request accepted and waiting for finish
Deployment instantiation
Notify about new deployment entry
Configure and start deployment process
Deployment finished
Notify about finished deployment
Update deployment entry status
Deployment finished callback
Notify about finished deployment
Notify about finished deployment
Show deployment finished

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 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 is 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 automation configured to use CSEM and then inspects the results from the perspective of the IaC automation tooling as well as CSEM.

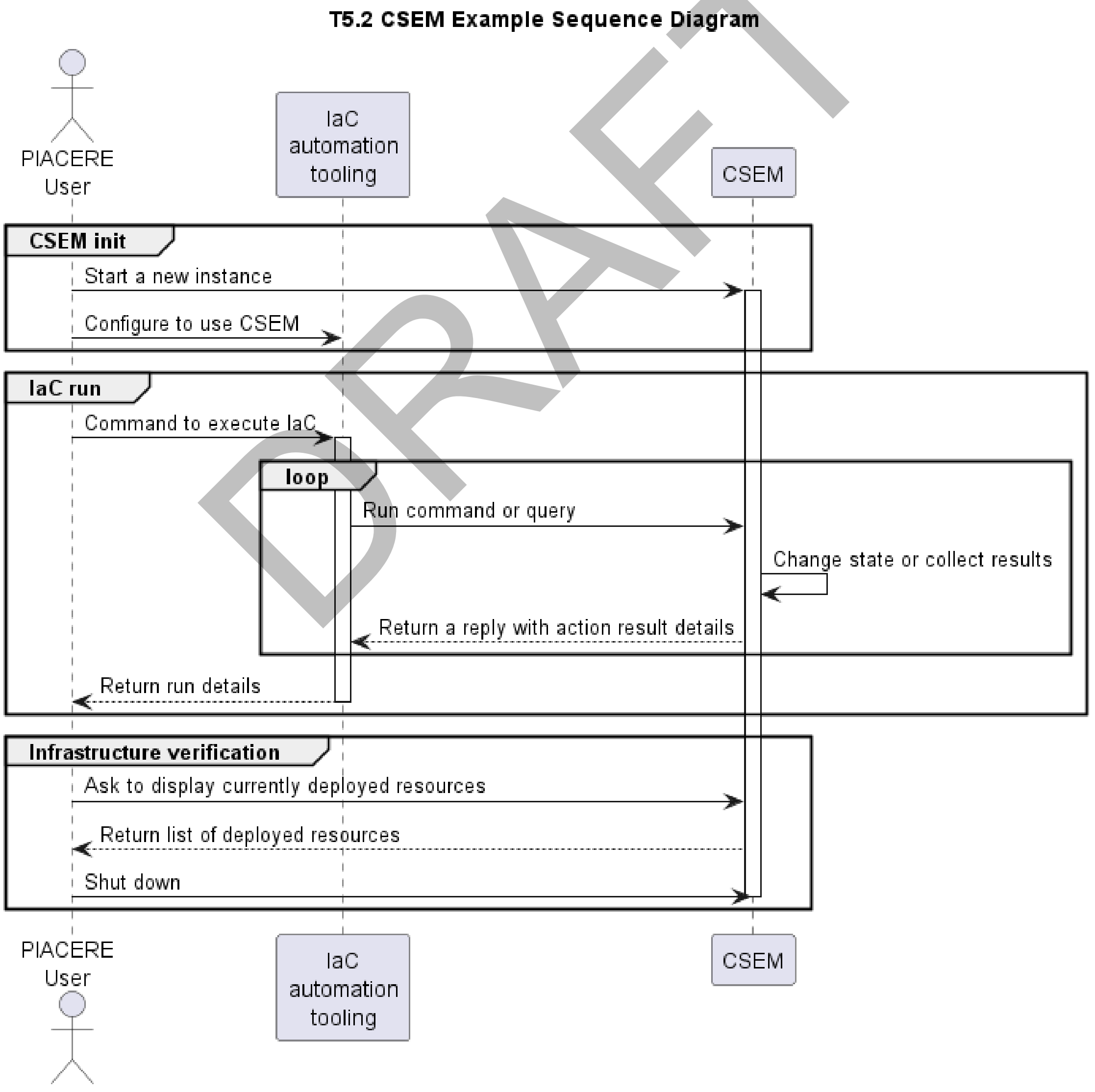
T5.2 CSEM Example Sequence Diagram
PIACERE User
IaC automation tooling
CSEM
CSEM init
Start a new instance
Configure to use CSEM
IaC run
Command to execute IaC
loop
Run command or query
Change state or collect results
Return a reply with action result details
Return run details
Infrastructure verification
Ask to display currently deployed resources
Return list of deployed resources
Shut down
PIACERE User
IaC automation tooling
CSEM

Figure 2 – Example CSEM Sequence Diagram

2.2.3 Fitting into the 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 (from D2.2) – a diagram provided by Work Package 2 that updates the one presented in version 1.

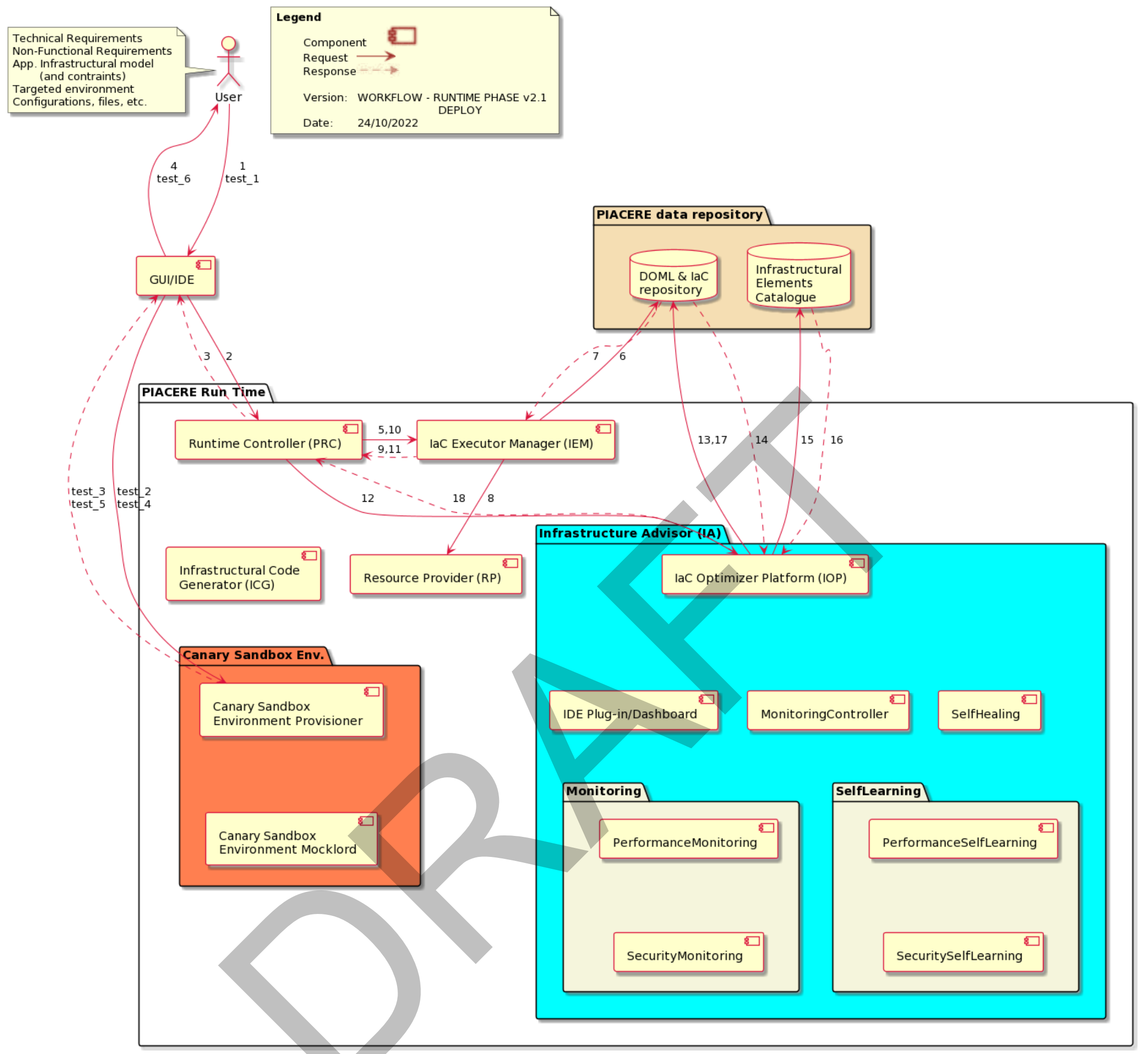

Figure 3 - PIACERE Runtime Diagram (from D2.2)

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).

2.3 Technical description

2.3.1 Canary Sandbox Environment Provisioner - CSEP

2.3.1.1 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.

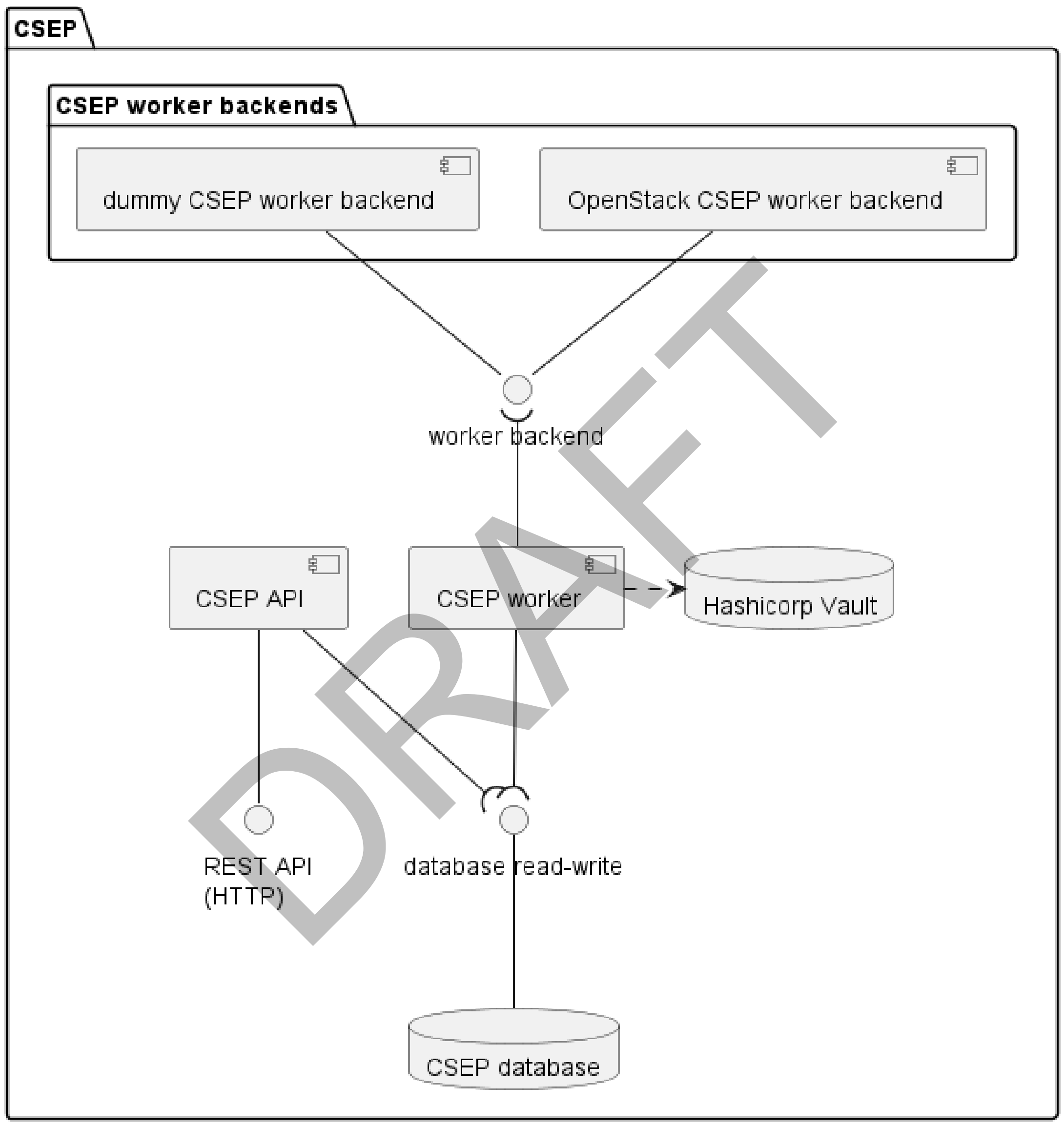

Figure 4 - CSEP Architecture Diagram

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.

2.3.1.2 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.

2.3.1.3 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 [4] 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 [5] 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 [6]. FastAPI relies heavily on Python's type annotations to drive API definition and its validation. FastAPI also offers native generation of OpenAPI specification [7] and is capable of rendering docs in Swagger UI [8] as well as Redoc [9]. Of indirect dependencies, a notable one is Pydantic [10]. 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 [11] 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 [12]. 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.

2.3.2 Canary Sandbox Environment Mocklord – CSEM

2.3.2.1 Prototype architecture

The architecture of CSEM is based on the moto [13] 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.

2.3.2.2 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.

2.3.2.3 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 [14] 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.

2.3.2.3.1 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.

2.3.2.3.2 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.

2.3.2.3.3 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.

2.3.2.3.4 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.

2.3.2.3.5 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.

3 Delivery and usage

3.1 Changes in v2

Regarding delivery and usage, similarly to the implementation section, the biggest addition is the introduction of CSEM which is now properly documented using the same set of sections like CSEP. For CSEP, we have overall enhanced the user manual, refreshed it to match the current API and added the details on the integration with Hashicorp Vault.

3.2 Canary Sandbox Environment Provisioner - CSEP

3.2.1 Package information

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

- .bandit – a file to configure one of the code linters (bandit [15]),
- .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 [16] 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 [17] 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) [18] 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 [19] 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 [20] 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,
- tests – hold files relevant to testing, including unit tests and examples,
- tools – miscellaneous tools, mostly useful during development.

3.2.2 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 [21] (19.03+) and docker-compose [22] (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).

3.2.3 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 Postman[2] for more flexibility. There are 8 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 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 3 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” – however, we have already received a request for this functionality and it shall be implemented in the next release,

[2] https://www.postman.com/

- `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.

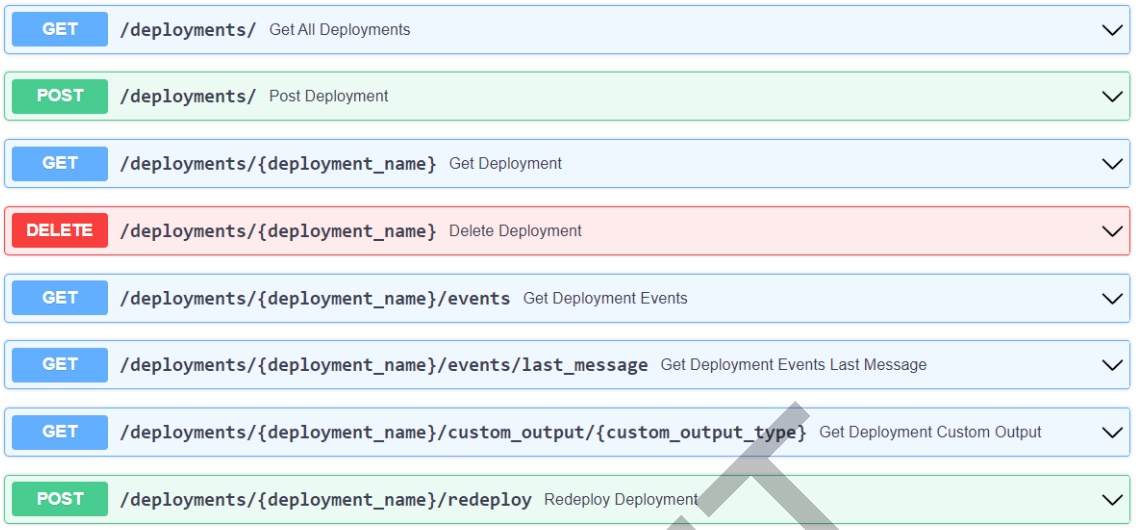
GET /deployments/ Get All Deployments
POST /deployments/ Post Deployment
GET /deployments/{deployment_name} Get Deployment
DELETE /deployments/{deployment_name} Delete Deployment
GET /deployments/{deployment_name}/events Get Deployment Events
GET /deployments/{deployment_name}/events/last_message Get Deployment Events Last Message
GET /deployments/{deployment_name}/custom_output/{custom_output_type} Get Deployment Custom Output
POST /deployments/{deployment_name}/redeploy Redeploy 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 [23] 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)

3.2.4 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):

- etcd3 (python library) – Apache 2.0
- etcd3 (the database service) – Apache 2.0
- FastAPI – MIT
- hvac – Apache 2.0
- Pydantic – MIT
- PyYAML – MIT
- Uvicorn – 3-clause BSD
- Vault – MPL 2.0

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

3.2.5 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/csep

3.3 Canary Sandbox Environment Mocklord - CSEM

3.3.1 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 [16] 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.

3.3.2 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 [21] (19.03+) and docker-compose [22] (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).

3.3.3 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 help[3]. For reproducible runs, we recommend 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).

[3] http://docs.getmoto.org/en/latest/docs/configuration/recorder/index.html

3.3.4 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.

3.3.5 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.

4 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 context, architecture, implementation, means of obtaining, and usage of the prototypes for M24 were explained in detail.

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. The main innovation was the enablement of testing of AWS-compatible IaC with enhanced validation to ensure correctness in the desired scope. Another minor innovation was the improvement of CSEP to better realise its role – via a more user-friendly API and the integration with Hashicorp Vault.

The remaining, final version of this deliverable, v3, at M30 will focus on implementing adjustments requested by the use cases (such as the “undeployment”) and extending and polishing the existing documentation and its mapping between scenarios.

5 Annexes

5.1 State of the art (SOTA) from D5.4 (v1)

This section is copied verbatim from D5.4 to make this document more self-contained.

5.1.1 Related existing tools

The way different companies and their DevSecOps specialists approach the CSE idea differs from case to case. It depends on multiple factors, such as company's and specialists' experience but also the scope and object of testing. For the purpose of this document, CSE means an environment where the IaC can be tested dynamically without impacting the final (production) target environment. The CSE idea is not adopted very widely, so the number of available solutions is limited. In our review we have focused on existing and mature solutions, as they can be used as a reference for our work.

In recent years, Hashicorp's Vagrant [24] has gained popularity in the DevOps environment, especially focused on the Dev part of DevOps. Vagrant allows to use a Domain Specific Language (DSL) in Ruby language to define a setup of VMs, their networking and storage. The DSL also allows to easily plug-in VM configuration steps via a chosen tool, e.g., Ansible or Puppet (or even just shell script). Vagrant supports multiple providers, such as a local VirtualBox or a local VMware solution. One of the prominent features of Vagrant that accounts for its popularity is the system of boxes. Vagrant box is a way to deliver the VM image. Hashicorp has provided some official ones, mostly to kickstart the project, but users are able to create their own. Hashicorp also provides a public space to share the boxes and thus there is an abundance of available boxes which cover many operating systems and even preinstalled solutions, such as LAMP (Linux-Apache-MySQL-PHP). Recently, Vagrant has also incorporated Docker support but it is not the major goal of the project.

While Vagrant focuses on VMs, another solution relevant to CSE, LocalStack [25], focuses on the serverless approach (FaaS – Function as a Service). Currently, only AWS (Amazon Web Services) are supported. The obvious problem with serverless/FaaS is that the solutions are highly dependent on the providers. The project chose AWS because of its popularity in this market segment. LocalStack is based on Python's moto [26] library which aims to model the AWS API and provide a mocked-up experience, mostly for testing code interacting with AWS.

Finally, companies often turn to ad-hoc solutions, such as providing an on-premise cloud based on, e.g., OpenStack, and then running their custom IaC. Similarly, at a different level, specialists boot ad-hoc Docker Swarm and Kubernetes clusters.

5.1.2 Analysis of shortcomings

While the different solutions appeal to their users, the issue is with their generality, scalability and manageability. Both the popular Vagrant and LocalStack are limited to localhost and focus on the development efforts, not the ops side of DevOps. Vagrant is mostly VMs (and perhaps containers) with minimal support for storage and networking options (which does not surprise considering it's a localhost solution). LocalStack is AWS Lambda for those who want to test their AWS integrations locally. Ad-hoc solutions come with the obvious problem of manageability and lack of standardisation. Often, within one company, two such ad-hoc solutions differ enough to cause issues for solution integrators, not to mention the actual costs of maintaining the setup. Finally, none of the mentioned solutions allow for simulation of the outcome, thus, they always cost considerable resources proportional to the complexity of the tested IaC.

6 References

[1] 'Vault'. HashiCorp, Nov. 14, 2022. Accessed: Nov. 14, 2022. [Online]. Available: https://github.com/hashicorp/vault

[2] Hashicorp, 'Terraform by HashiCorp'. https://www.terraform.io/ (accessed Mar. 29, 2021).

[3] 'ansible/ansible'. Ansible, Feb. 18, 2022. Accessed: Feb. 18, 2022. [Online]. Available: https://github.com/ansible/ansible

[4] 'etcd'. etcd-io, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/etcd-io/etcd

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

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

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

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

[9] 'redoc'. Redocly, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/Redocly/redoc

[10] S. Colvin, 'pydantic'. Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/samuelcolvin/pydantic

[11] 'hvac'. hvac, Nov. 12, 2022. Accessed: Nov. 14, 2022. [Online]. Available: https://github.com/hvac/hvac

[12] 'openstack/kolla-ansible'. OpenStack, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://opendev.org/openstack/kolla-ansible

[13] S. Pulec, 'spulec/moto'. Mar. 27, 2021. Accessed: Mar. 21, 2021. [Online]. Available: https://github.com/spulec/moto

[14] 'Boto3 - The AWS SDK for Python'. the boto project, Nov. 13, 2022. Accessed: Nov. 14, 2022. [Online]. Available: https://github.com/boto/boto3

[15] 'PyCQA/bandit'. Python Code Quality Authority, Nov. 13, 2022. Accessed: Nov. 15, 2022. [Online]. Available: https://github.com/PyCQA/bandit

[16] 'GitLab CI/CD | GitLab'. https://docs.gitlab.com/ee/ci/ (accessed Nov. 15, 2022).

[17] 'Mypy: Static Typing for Python'. Python, Nov. 15, 2022. Accessed: Nov. 15, 2022. [Online]. Available: https://github.com/python/mypy

[18] 'PDM'. PDM, Nov. 15, 2022. Accessed: Nov. 15, 2022. [Online]. Available: https://github.com/pdm-project/pdm

[19] 'tox automation project'. tox development team, Nov. 15, 2022. Accessed: Nov. 15, 2022. [Online]. Available: https://github.com/tox-dev/tox

[20] 'Gherkin'. Cucumber, Nov. 15, 2022. Accessed: Nov. 15, 2022. [Online]. Available: https://github.com/cucumber/gherkin

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

[22] 'Docker Compose'. Docker, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/docker/compose

[23] 'Kubernetes API Reference', *Kubernetes*. https://kubernetes.io/docs/reference/ (accessed Nov. 09, 2021).

[24] 'Vagrant'. HashiCorp, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/hashicorp/vagrant

[25] 'LocalStack'. LocalStack, Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/localstack/localstack

[26] S. Pulec, 'Moto - Mock AWS Services'. Nov. 09, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/spulec/moto