Deliverable D5.4

Canary Sandbox Environment prototype – v1

Editor(s):	Radosław Piliszek (7bulls)
Responsible Partner:	7bulls
Status-Version:	Final – v1.0
Date:	25.11.2021
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	30.11.2021

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:	The 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.
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	17.02.2021	First draft version.	Radosław Piliszek (7bulls)
v0.8	09.11.2021	Proposed version for Y1.	Radosław Piliszek (7bulls)
v0.9	25.11.2021	Fixes after the internal review.	Radosław Piliszek (7bulls)
V1.0	29.11.2021	Ready for submission	Leire Orue-Echevarria (TECNALIA)

Table of contents

List of tables

List of figures

Terms and abbreviations

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 (a future tool, related to lightweight, simulated approach)
CSEP	CSE Provisioner (the proposed tool, responsible for target environment provisioning)
CSP	Cloud Service Provider
DevOps	Development and Operations
DevSecOps	Development, Security and Operations
DoA	Description of Action
EC	European Commission
GA	Grant Agreement to the project
HW	Hardware
IaC	Infrastructure as Code
IEM	IaC Execution Manager (one of the tools built in the PIACERE project, delivered as part of Task 5.1)
KPI	Key Performance Indicator
KR	Key Result
KR8	Key Result 8 (which this deliverable is about)
SW	Software
TSR	Technical Specification Report (e.g., this document)

Executive Summary

This document is the Technical Specification Report (TSR) of the first Canary Sandbox Environment tooling prototype developed in the PIACERE project. The Canary Sandbox Environment 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 part of the effort of Task 5.2 in Work Package 5.

The document starts by describing the State of the Art (SOTA) in the related field of provisioning sandbox environments and puts the proposed solution in a proper context. Then it moves to describe the goals and non-goals of the Canary Sandbox Environment. Following the introduction, the section on implementation describes the architecture of the proposed tooling and its realisation. In the next section, it 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 first revision, enables users to provision one of the first target environments – OpenStack – that can be utilised by the IaC Execution Manager (IEM) when running IaC to achieve sandboxing understood as separation from the production environment. This accompanying TSR focuses on the description of the CSE Provisioner (CSEP) which is the tool enabling the deployment of OpenStack on users' demand. CSEP's motivation, architecture and technical aspects are discussed in detail. Moreover, ground is laid for future work.

Future versions of the Canary Sandbox Environment tooling will cover more requirements and will be described in the future versions of this deliverable – D5.5 (v2; due month 24) and D5.6 (v3; due month 30). These next versions will include extensions in the target environments supported as well as research on more lightweight, simulated approach.

1 Introduction

1.1 About this deliverable

This document is the Technical Specification Report (TSR) of the first 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 first version, one of the proposed tools, Canary Sandbox Environment Provisioner (CSEP), is described in detail. CSEP is used to provision Canary Resource Providers (CRPs); in this version only OpenStack is offered but the implementation is extensible. 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 6 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 State of the Art (SOTA) in the context of similar solutions. The 3rd section describes the architecture of the proposed solution. The 4th section details the steps to obtain and consume the software. The 5th section concludes the document. The 6th section contains the references cited within the TSR.

2 Canary Sandbox Environment context

2.1 Current state of art

2.1.1 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 [1] 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 [2], 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 [3] 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.

2.1.2 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.

2.2 PIACERE Canary Sandbox Environment goals

PIACERE CSE aspires to alleviate the issues with simple and ad-hoc solutions varying from specialist to specialist. The goal of CSE is to provide tools that would allow to dynamically test

the IaC in a fast and cheap manner. To this end, the detailed goals are to provide both real (non-simulated) and simulated deployment variants to satisfy the varying needs of dynamic IaC testing platforms. On one hand, solutions such as opinionated OpenStack deployment would cater for the real deployment variant – allowing to simply proceed with actual resources, being limited only by provided capacity. On the other, the use of mocks would be the basis for the simulated approach.

2.3 PIACERE Canary Sandbox Environment non-goals

It is not a goal of CSE to provide an environment for Non-Functional Requirements (NFRs) testing, such as performance testing, because the CSE cannot guarantee at any rate that the results will be relevant to the production environment. The only use of CSE in that regard would encompass comparing different realisations of IaC on the same CSE.

It's also not a goal of PIACERE to simulate multiple cloud providers out of the box nor provide complete coverage of their API services. The implementation, however, is forethought to be easily extensible so this could be something to be added later if such need arises from the users.

3 Implementation

3.1 Functional description

The proposed prototype is to offer two approaches to the CSE: to provide a real (non-simulated) Canary Resource Provider (CRP) and a simulated one. The CRP is to be used by IaC Execution Manager (IEC) – another tool from the PIACERE project – as a target when running IaC, replacing the actual, production resource provider. 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 initial set of planned supported CRPs is: OpenStack (for real [non-simulated] actions) and Canary Sandbox Environment Mocklord (CSEM; for simulation).

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.

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	However, this is yet to be tested on actual examples with IEM used (the executors run directly are fine).
REQ34 / WP5.2-REQ2	CSE to keep track of and allow querying of the deployment state to allow comparison against the expected one.	Implemented	The current CSEM API allows to query the status of the created deployment and its various details.
REQ37 / WP5.2-REQ3	CSE to have a simulated mode limited to provisioning.	Not implemented	This is planned for Y2.
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	OpenStack is the target platform for this requirement, and it is supported in this release.
REQ39 / WP5.2-REQ5	CSE to enable extensibility (documented way): adding new mocked services, adding new "real" deployments.	Partially implemented	The currently provided implementation of CSEM allows for easy extension in terms of supported types of deployments. The documentation on that is pending and the point about mocked services relies on REQ33. Y2 is the milestone to finish this.

[1] Reported in D2.1

The main innovation developed in this cycle is the opinionated deployment of OpenStack that is accessible via an easy-to-use REST API. The pending innovation is the provider of mocked-up variants of public cloud providers services, targeting AWS as the example platform.

There are two tools being provided as part of CSE – Canary Sandbox Environment Provisioner (CSEP) and Canary Sandbox Environment Mocklord (CSEM). Their functionality and purpose are explained in the following two subsections.

3.1.1 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 desired platform. The main sequence diagram is presented in Figure 1 - CSEP Main Sequence Diagram.

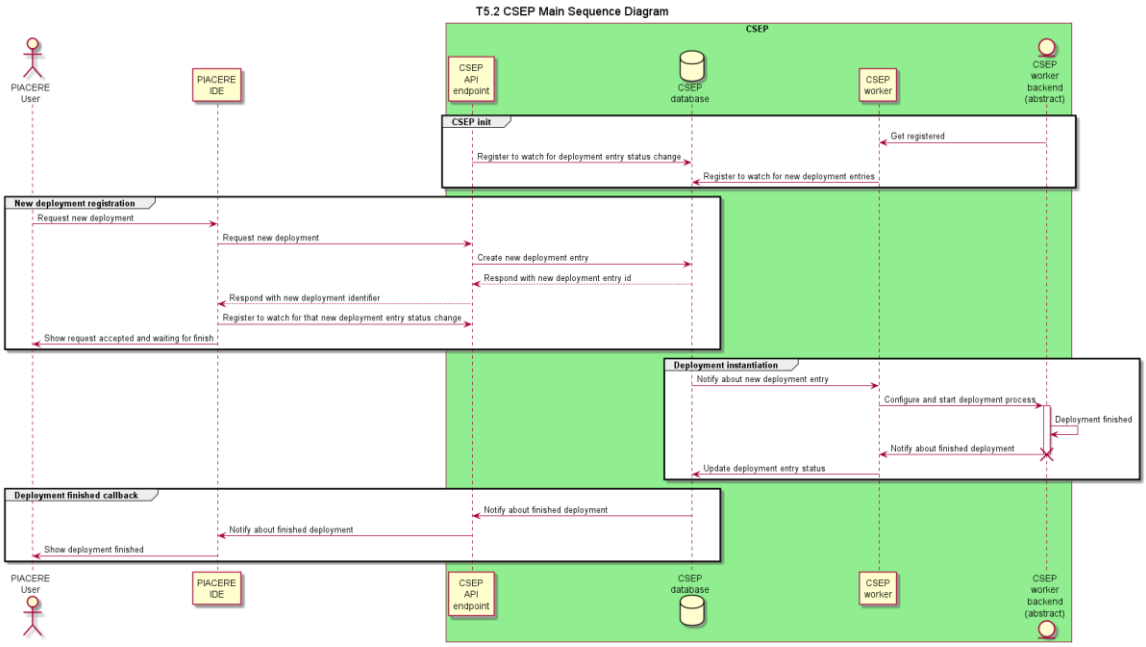

Figure 1 - CSEP Main Sequence Diagram

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.

3.1.2 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 plan is to research the usefulness of such approach to dynamic IaC testing. The future prototype will target a subset of AWS APIs. CSEM is assumed to have much lower cost compared to real (non-simulated) resource providers. Due to simulation, this variant of Canary Resource Provider will allow 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 will not be possible to connect to the "deployed" virtual machine as there will be none to target.

3.1.3 Fitting into overall PIACERE Architecture

The prototype of 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 to be our modular implementation of mocked-up resources provider.

CSE tools live in the runtime phase of the PIACERE framework. They have been depicted at the bottom of Figure 2 - PIACERE Runtime Diagram (version 1.5) – a diagram provided by Work Package 2. The figure shows the usage of CSEP by IDE to provide a (canary) resource provider that IEM can use.

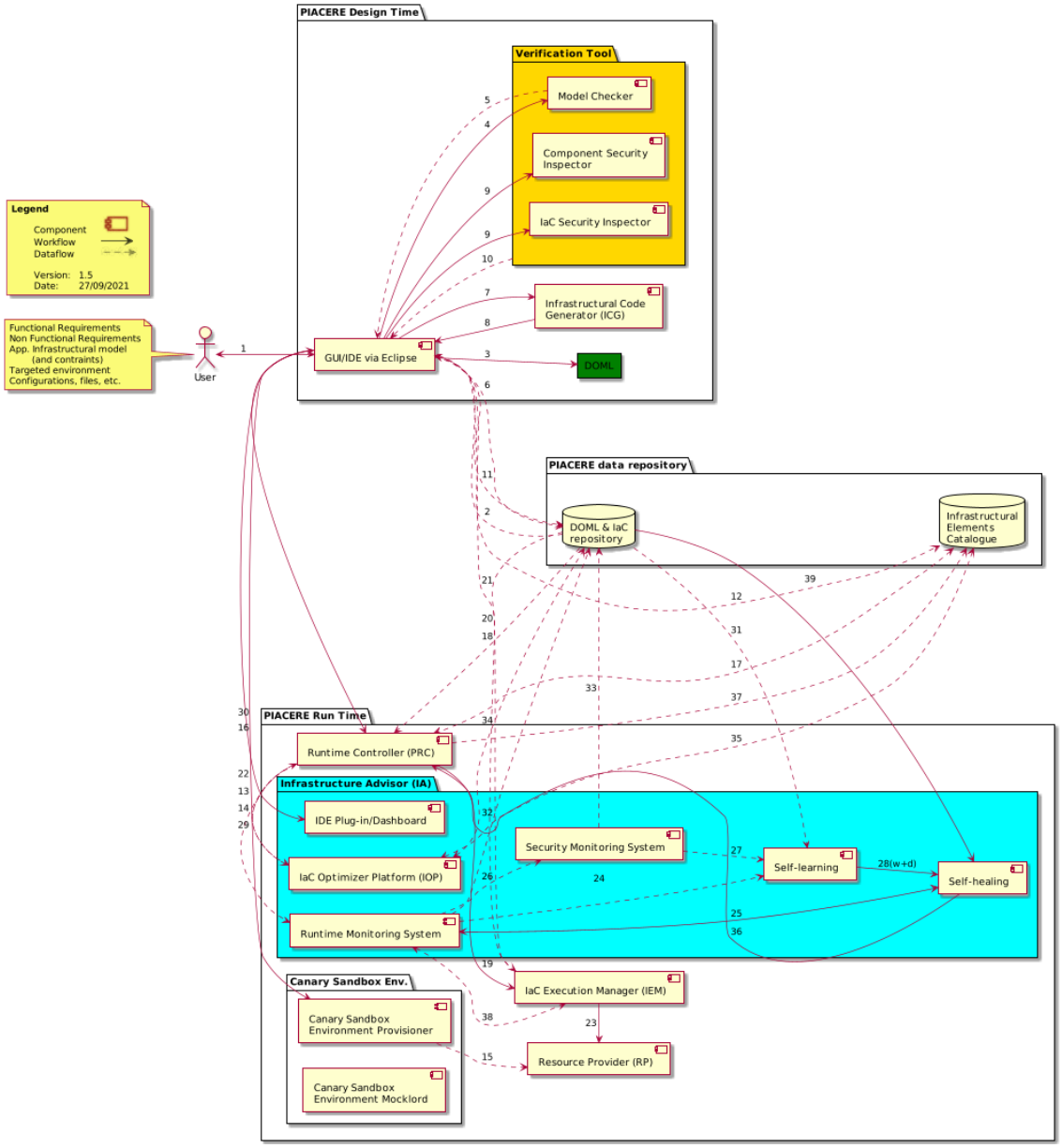

Figure 2 - PIACERE Runtime Diagram (version 1.5)

3.2 Technical description

3.2.1 Canary Sandbox Environment Provisioner - CSEP

3.2.1.1 Prototype architecture

The prototype architecture is shown in Figure 3 - 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.

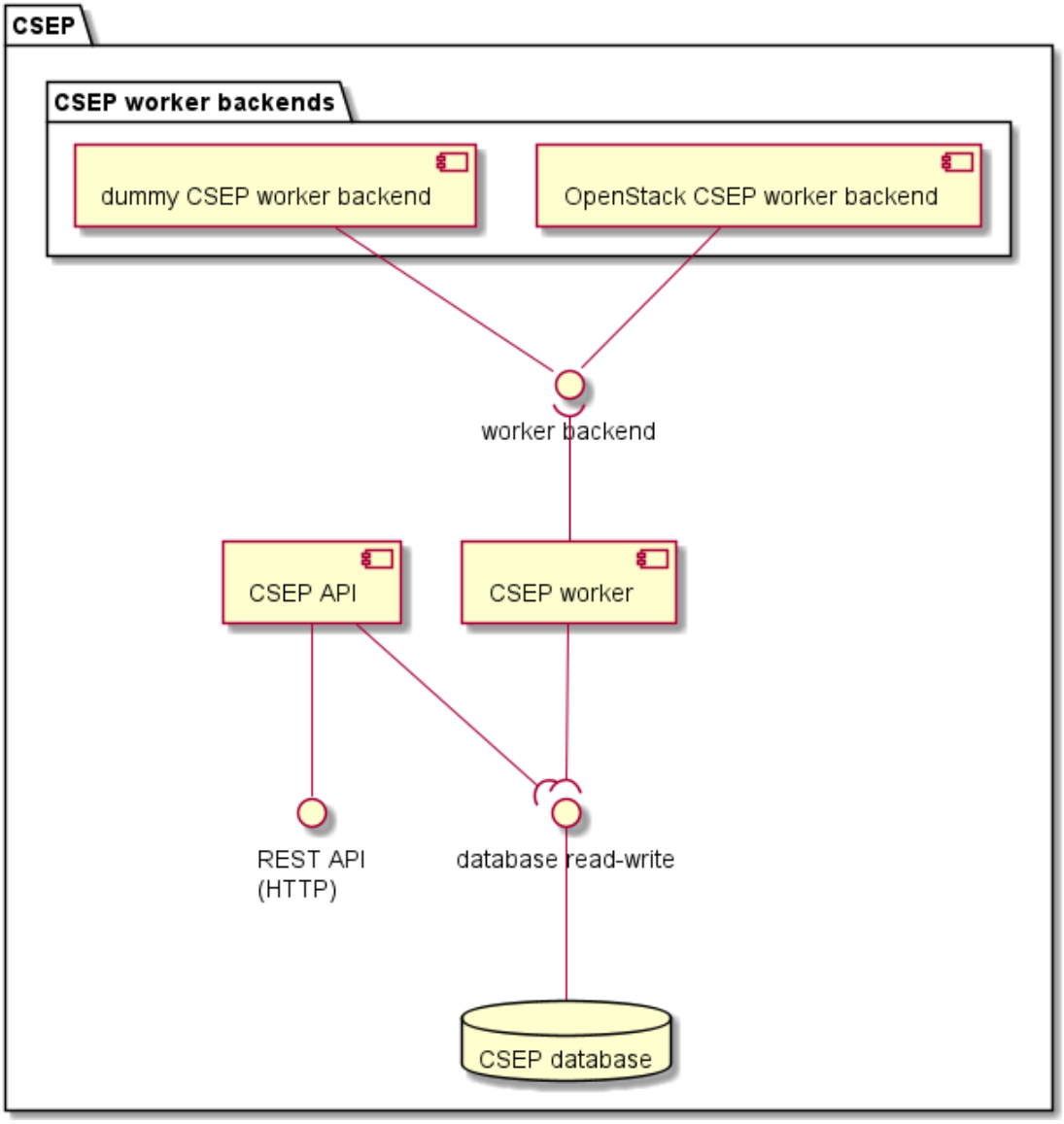

Figure 3 - CSEP Architecture Diagram

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.

3.2.1.2 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. In this version, two backends are provided: dummy (for integration testing purposes) and OpenStack.

3.2.1.3 Technical specifications

The prototype is written in Python and tested specifically with Python 3.10. It should, however, work with any modern Python (3.8+) as no bleeding edge features of the language are used. Nonetheless, the exact version is not important as the prototype is prepared to be developed

and deployed in a containerised manner which eliminates the need to ensure that the underlying platform has the correct Python version nor any other dependencies.

Both provided components utilise the etcd [4] database, currently in version 3.5, and thus also the accompanying library (etcd3) to connect to it, currently in version 0.12. 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, currently in version 0.70. 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], currently in version 0.15. 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] – its version is pinned by FastAPI. Pydantic offers the framework for extra type annotations and the actual validation in instantiated objects.

The worker component does not currently utilise any additional libraries, it reuses the Pydantic library used in the API component for easy serialisation-deserialisation from the common database. However, 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], currently of the Xena release. Kolla Ansible is a community-powered OpenStack deployment project which utilises Docker (and soon podman) 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.

3.2.2 Canary Sandbox Environment Mocklord - CSEM

In this deliverable the CSEM is not provided and thus the technical description for it is not available. This section is mentioned for completeness.

4 Delivery and usage

4.1 Canary Sandbox Environment Provisioner - CSEP

4.1.1 Package information

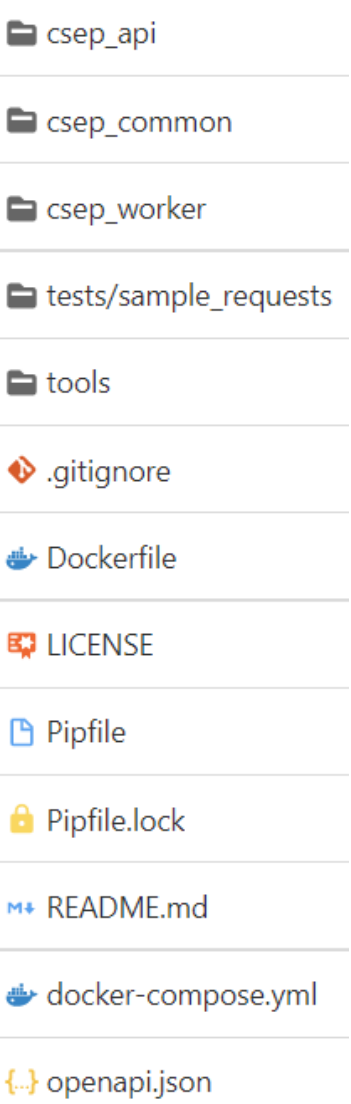

Figure 4 - Package contents

The package contents are displayed (as in GitLab) in Figure 4 - Package contents. There are 5 directories and 8 files (13 entries in total). The files are as following:

- .gitignore – a file relevant to Git (repository) operations, containing file paths to ignore,
- Dockerfile – a file containing the recipe to build the container image which supports the software,
- LICENSE – a placeholder file for the software license (to be established),
- Pipfile – a file with Python dependencies (consumed by pipenv [12])
- Pipfile.lock – an accompanying file for Pipfile, it locks versions to ensure repeatable builds,
- 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,
- openapi.json – a JSON file with OpenAPI specification dump from the current version of software,

and the directories are:

- 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,
- tests (currently only tests/sample_requests) – hold files relevant to testing (currently example sample requests),
- tools – miscellaneous tools, mostly useful during development.

4.1.2 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 [13] (19.03+) and docker-compose [14] (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 3 containers corresponding to the two components and the database.

4.1.3 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 Postman[2] for more flexibility. There are four endpoints in total – two 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),

and two 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.

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

An example request body is 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 [15] 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.

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

```
{
    "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)

4.1.4 Licensing information

The licenses of software components produced as part of T5.2 are yet to be discussed. In future versions of the deliverable this will be clarified and reported.

4.1.5 Download

The code is available for review at Tecnalia's GitLab, in the private part of the PIACERE project: https://git.code.tecnalia.com/piacere/private/t52-canary-provisioner

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

4.2 Canary Sandbox Environment Mocklord - CSEM

In this deliverable the CSEM is not provided and thus the "delivery and usage" section for it is not available. This section is mentioned for completeness.

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 context, architecture, implementation, means of obtaining, and usage of the first version of the prototype were explained in detail.

The currently provided prototype offers the provisioning functionality via Canary Sandbox Environment Provisioner (CSEP) which utilises Kolla Ansible to deploy opinionated OpenStack using Ansible playbooks and Docker containers. Another main innovation, apart from the opinionated out-of-the box experience with OpenStack, lies in the easy-to-use, uniform REST API to request deployments and the extensible framework to add other deployment types to the one already implemented.

A future version of this deliverable will focus more on the simulated approach to CSE to provide a prototype of Canary Sandbox Environment Mocklord (CSEM). CSEM is to provide simulation of an existing cloud provider such that cloud-specific IaC can be tested non-invasively and with no additional costs, albeit without the possibility to deploy applications as no real resources are allocated.

6 References

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

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

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

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

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

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

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

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

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

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

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

[12] *Pipenv: Python Development Workflow for Humans*. Python Packaging Authority, 2021. Accessed: Nov. 09, 2021. [Online]. Available: https://github.com/pypa/pipenv

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

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

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