PIACERE



# Deliverable D4.5

# IaC Code security and components security inspection v2

| | |
|---|---|
| **Editor(s):** | Matija Cankar |
| **Responsible Partner:** | XLAB d.o.o. |
| **Status-Version:** | 1.0 Final Version |
| **Date:** | 28.11.2022 |
| **Distribution level (CO, PU):** | Public |

| | |
|---|---|
| **Project Number:** | 101000162 |
| **Project Title:** | PIACERE |

| | |
|---|---|
| **Title of Deliverable:** | IaC Code security and components security inspection v2 |
| **Due Date of Delivery to the EC** | 30.11.2022 |

| | |
|---|---|
| **Workpackage responsible for the Deliverable:** | WP4 - Verify the trustworthiness of Infrastructure as a code |
| **Editor(s):** | Matija Cankar (XLAB) |
| **Contributor(s):** | Anže Luzar (XLAB), Matija Cankar (XLAB), Nenad Petrović (XLAB) |
| **Reviewer(s):** | Juncal Alonso (TECNALIA) |
| **Approved by:** | All Partners |
| **Recommended/mandatory readers:** | WP2, WP3, WP5, WP8 |

| | |
|---|---|
| **Abstract:** | This deliverable will present the outcome of Task T4.2 and Task T4.3. The deliverable comprises both a software prototype [KR6-KR7] and a Technical Specification Report. The document will include the Security Inspector technical design and implementation aspects. The document will also include the Security Inspector technical design and implementation aspects |
| **Keyword List:** | IaC, SAST, IaC Security, DevOps, DevSecOps |
| **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 | 21.08.202x | First version of ToC | XLAB |
| v0.2 | 18.10.2022 | First draft | XLAB |
| v0.3 | 21.11.2022 | Version for internal review | XLAB |
| v0.4 | 23.11.2022 | Comments and suggestions received by consortium partners | Tecnalia |
| v0.5 | 25.11.2022 | Final version | XLAB |
| v1.0 | 28.11.2022 | Ready for submission | Tecnalia |

# Table of contents



# List of tables

# List of figures

# Terms and abbreviations

| | |
|---|---|
| API | Application Programming Interface |
| ASAP | As soon as possible |
| CI | Continuous integration |
| CVE | Common Vulnerabilities and Exposures |
| CSAR | Cloud Service Archive |
| CSP | Cloud Service Provider |
| DB | Database |
| DevOps | Development and Operations |
| DevSecOps | Development, Security and Operations |
| DoA | Description of Action |
| EC | European Commission |
| GA | Grant Agreement to the project |
| GUI | Graphical User Interface |
| HCL | HashiCorp Configuration Language (Terraform HCL) |
| IaC | Infrastructure as Code |
| IEP | IaC execution platform |
| IOP | IaC Optimization |
| JSON | JavaScript Object Notation |
| KPI | Key Performance Indicator |
| SW | Software |
| SAST | Static Application Security Testing |
| SCA | Software Composition Analysis |
| SotA | State of the Art |
| TOSCA | Topology and Orchestration Specification for Cloud Applications |
| UI | User Interface |

# Executive Summary

This Is a second technical deliverable from a series of three, describing the progress and plans of the IaC Security Inspector and Component security inspector (KR6 and KR7). The mentioned results are output of tasks T4.2 and T4.3 from the WP4.

This deliverable shows the progress of the IaC Security Inspector and Component security inspector (KR6 and KR7) during the second year of PIACERE project. In this document, the components are referred to as a single entity merging them together, called *IaC Scan Runner*. The focus is on new features and capabilities, emphasizing the novelties with respect to previous year, that we describe as innovations.

The first innovation focuses on improved scan result aggregation, together with their ranking according to their outcome and persistence for later browsing. This means that user can receives more informational result and ordered by specific ranking.

The second innovation focuses on speedup and efficiency. For that, a concept of compatibility matrix was introduced enabling the execution of relevant checks only within the scan workflow, so the overall processing time is reduced as checks for non-existent file types are skipped. Moreover, the aspects of scan-related preferences are considered with main aim to enable re-use in case of similar archives.

The goal of presented two innovations is to both improve the response time of overall PIACERE IDE workflow, but also to make it more user-friendly by giving the extended result summary and ability to browse the results in more convenient way. Finally, some elements related to extendibility with new tools and providing support for novel IaC standards are considered, but mainly left for future work until the end of the project.

The mission of this work package remains the same – find and spot as many IaC issues as possible that could affect the deployed application.

# 1 Introduction

This document represents the second technical deliverable out of three which are planned, summarizing the current progress and future related to the development of IaC Scan Runner (KR6 and KR7). The results presented in this document are the outputs of tasks T4.2 and T4.3 from the WP4.

In the first year we focused mostly on IaC Security Inspector and integration part. Therefore, in the second year we focused more on IaC Component Inspector and introducing an updated version of IaC Scan Runner, combining results of both inspectors, with persistence layer which focuses on enhancement of the following aspects:

- more sophisticated scan result information summarization and visualization
- storage of scan results and later browsing them
- keeping scan configuration parameters and enable its re-use for further scans
- organizing scan tasks within projects to reduce time needed for configuration
- reduce the scan execution time by avoiding non-compatible file types
- improve the extendibility allowing to add new check tools to the scan workflow
- new scans integrated
- component scanning expanded to Ansible Collections

However, it would be still possible to use IaC Scan Runner in a stateless manner (without relying on persistence layer), in a way already described within the previous document [1]. In that case, specific set of settings (as will be described later in this document) has to be applied in order to ensure backwards compatibility.

## 1.1 About this deliverable

The main aim of this deliverable is to explain the new usage scenarios and improvements within the second version of IaC Scan Runner, enabled thanks to introduction of persistence layer. The main parts targeted to be improved, compared to the previous version will be mentioned. First, we want to make the obtained results more readable to humans and easily retrievable even later in time. On the other side, as it might happen that particular user would like to perform a set of scans with the same settings against multiple IaC archives of the same type, then we also have goal to enable re-use of scan tool configuration in order to reduce the time needed for manual settings. Finally, the reduction of execution time required for huge IaC archives is aimed to be achieved by avoiding calling scan workflow tools that target files that are not actually present within the provided IaC.

## 1.2 Document structure

The rest of this document has the following structure:

- Section 1 presents an overall description of the delivery and its main goal is provided.
- Section 2 focuses on the implementation of the improvements with repsect to the previous version of the tool. In the Section 2.1 *Approach* discuss how we decide which checks to cover.
- Section 3 describes the delivery and usage of the developed tools.
- Section 4 lists the conclusions
- Section 5 lists references and citations used throught the document, and
- Appendix: The details of IaC Scan Runner database schema.

# 2 Implementation

The main purpose of the delivered prototype remains the same as described in the previous version of the deliverable. It aims to help DevOps Engineers minimising the potential issues with using their code, which should be done in two main steps. The first one is execution of static application security testing (SAST) tools against the IaC code which performs analysis before the deployment. On the other side, the second is creation of live tests and specialised monitoring that finds issues during the application lifecycle. In this context, our work focuses on providing tools where SAST tool management can be applied on the IaC cases with ease.

This document summarizes the implementation until the second year. All the functionalities from the first year [1] are completed, while the focus here is on the new extensions introduced in the second year, aiming to cover the following aspects with goal to improve the usability of the tool and overall user experience:

- Visual result summary and scan result prioritization
- Browsing the results of previously performed scans
- Reduce time needed for configuration by introducing re-usable scan configurations
- Improve scanning speed by performing only relevant checks for detected file types

The features, apart from scan configuration management are fully implemented. On the other side, result processing and scan compatibility are expected to be extended in future if support for new checks has to be included.

## 2.1 Approach

We follow the same approach as it was planned in D4.4 [1], this means that, primarily, the needs of stakeholders responsible for PIACERE use cases were taken into account regarding the process of check tool selection. Together we prepared a list of checks and make decision of which could be included in IaC Scan Runners. Based on analysis of their usage scenarios, it was identified that the Terraform HCL, Ansible and docker (hadolint) are the must have. . Additionally we included all available open source checks that have som potential use PIACERE use cases or were planned already in the proposal stage – like TOSCA YAML.

The decision of why integrate other checks are:

- **Exploitation**: if we cover more checks, this means user can have one service for multiple languages and project. As PIACERE framework is meant to be exploited also per-partes, we know that this provides a new potential edge to the IaC Scan Runner as it can be user to cover all other potential checks.
- **Future-proof result**: PIACERE framework is not limited only to Terraform, Ansible and Docker. DOML is not language specific, therefore, in future it could support any other language, e.g. python or shell.
- **Other *support* languages**. Some checks do not target directly the IaC languages, but the snippets or pieces of code that frequently can become a part of the IaC. This is the reason that we use also HTML or JavaScript, as both are very popular in web technologies.
- **Gap analysis**: Most of the IaC languages have stable progress and updates, except Ansible, which is very promising, but involved an important architectural change in past – most of the non-core functionalities it is covered by so-called *Collections*, that are Ansible libraries, puggable components that enrich Ansible capabilities. The Ansible Collecitons are do not have a specific check, therefore we saw the opportunity to fill this gap – use cases in PIACERE need that and the community is very eager to have this kind of tool. The need for this kind of tool was demonstrated also on

AnsibleFest – an industrial event in US where XLAB contributed with specific talks and workshops.

From that reasoning we compiled a final list of the checks which is available in Table 3.

## 2.2 Changes in v2

As already introduced in introduction, a various set of changes were made during the second iteration of development. They fall into the following categories:

**Extension of the outputs and responses**: When it comes to scan result outputs, both JSON and HTML results were extended with additional information that would ease their further usage. First, we process individual logs in order to determine the outcome of the check – passed or failed (in case that problems were detected). Additionally, HTML result summary was extended, so the outcome of checks affects the way they are visualized in order to cover the aspects of scan result prioritization. On the other side, auxiliary info, such as timestamp and execution duration is also added to the results.

**Extended coverage of scan checks**: New checks were added to fulfil the planned support presented in a table in D4.4. Besides, the full report for result summarization and prioritization is currently available for the checks, which is discussed within compatibility matrix-related subsection.

**Result persistency and sub-query**: to improve scan result management and configuration we introduced the persistence layer, which enables to temporarily store results and gives ability to browse them later. Moreover, another novelty is the ability to organize scans within projects, which will share the common configuration and save the user's time when it comes to scans of similar IaC archives.

**Describing alternative scenarios and demonstrate features**: A Gherkin/Cucumber notation was used to describe all possible scenario workflows of the IaC Scan runner and allow users to better understand it potential and allow to respond with feedback.

The changes were successfully integrated into the service, each detail will be described in the following sections.

## 2.3 Functional description

The functional description of the IaC Security tools was observed in threefold:

- Y1 requirements and feedback
- Y2 requirements
- Component features described in Gherkin/Cucumber notation.

The Y1 requirements (Table 1) were accomplished according to the project plan, except the requirement REQ80 was abandoned as it did not have a sense. Inspection is made over the whole IaC bundle regardless if the content is prepared for Canary environment or for any other staging (testing, production) environment.

*Table 1 Y1 requirements*

| REQ # | Description | Complexity /Task | Acceptance | Priority | Status |
|---|---|---|---|---|---|
| REQ23 | IaC Code Security Inspector must analyse IaC code | Medium | **ACCEPTED** | MUST HAVE | **Done** |

| | | | | | |
|---|---|---|---|---|---|
| | w.r.t. security issues of the modules used in the IaC. | T4.2 | | | |
| REQ24 | Security Components Inspector must analyse and rank components and their dependencies used in the IaC. | Medium<br>T4.3 | **ACCEPTED** | MUST HAVE | **Done, improvements in REQ106** |
| REQ65 | IaC Security Inspector and Component Security Inspector should hide specificities and technicalities of the current solutions in an integrated IDE. | Low<br>T4.2, T4.3, WP3 | **ACCEPTED** | MUST HAVE | **DONE** |
| REQ66 | IaC Code security inspector must provide an interface (CLI or REST API) to integrate with other tools or CI/CD workflows. | Medium<br>T4.2, T2.2 | **ACCEPTED** | MUST HAVE | **DONE**. |
| REQ67 | IaC Component security inspector must provide an interface (CLI or REST API) to integrate with other tools or CI/CD workflows. | Medium<br>T4.2<br>T2.2 | **ACCEPTED** | MUST HAVE | **DONE** |

After the integration of the Y1 results into the PIACERE environment we started productive conversations with use case providers, stakeholders and integrational team working on IDE. Together we set up the list of improvements and defined a set of new requirements presented in Table 2. Table shows the overview of new functional requirements for IaC Security Inspector and IaC Component Inspector.

*Table 2 Functional requirements*

| Req ID | Description | Status | Requirement Coverage at M24 |
|---|---|---|---|
| REQ106 | The organization of scan results should respect the scan outcome | Partially done, output summarization missing for some tools | Both JSON and HTML responses are updated, so results are organized into several different categories with respect to check outcome. |
| REQ108 | Management of scan results after the scan process is finished (supporting short term persistence) | Done | Scan results are stored in summarized form (JSON output) into persistence layer. |
| REQ109 | Scan configuration management | Partially done, initial REST API available | Added support for projects with user-defined configuration |

| | | | |
|---|---|---|---|
| | | | which can be re-used for multiple scan tasks. |

More in detail:

- REQ106**: Output of individual scan tools should be separated into different categories**, according to the actual outcome. In that context, we introduce the following categories: 1) **Problems** – check tool has detected issues 2) **Passed** – No problems were detected 3) **No files** – in case when particular check did not find relevant files to be scanned. Additionally, we also introduce one more category – Info, which refers to checks in wider sense that have different type of outcome, presenting useful information about the scanned IaC archive (such as number of files). On top of that, **HTML output** has been updated with visualization in tabular form, where checks that detected are shown first, while the others are placed below in the mentioned order (Passed, No files, Info). When it comes to completeness of this requirement implementation, summarization of output for some of the check tools is still work in progress.
- REQ108: **Output of scan in JSON form is entirely stored thanks to persistence layer**. Apart from check outcomes, some additional auxiliary info is included, such as name of IaC archive, timestamp when scan was performed and duration of scan task execution. Apart from that, basic functionalities **enabling visualization**, **filtering** and **deletion** of the scan results are also present. Finally, functionality enabling period deletion of old scan results is also included. All the planned activities related to implementation of this requirement are completed.
- REQ109: **Scan management configuration.** Scans are organized into projects with common configuration that includes both the list of enabled checks and configuration parameters required for some of the individual tools. The main goal of this requirement is **to reduce the time** needed for manual **scan configuration** set up each time scan is performed thanks to persistence and re-use of already created **user-defined configurations**. Additionally, simple API enabling filtering by project id and deletion of projects is also included. While all the basic capabilities related to scan project management have been already implemented, tool configuration-related functionalities are still in progress. Despite that the list of enabled tools can be stored, the persisted parameters (such as secrets, ids or tokens) are still not leveraged for **configuration of individual checks**.

To better understand the needs of the users and discuss the requirements, we prepared the scenarios in Gherkin/Cucumber notation. The notation allows very simple and plain descriptions of the application use and demonstrate it's potential. Beside the easier conversation with the stakeholders, the notation will be a basic cornerstone for the QA team to set acceptance criteria and help use-cases to set the validation scenarios. The IaC Scan Runner features are presented in the following listing.

```
Feature:  IaC Scan Runner - scan project creation, configuration and
re-use

   As a PIACERE user I want to create a new scan project with common
configuration for further reuse across multiple IaC archive scan runs

Scenario: Create new scan project (KR6 and KR7)
Given A generated IaC code from DOML
When user navigates to the IaC document/zip
And right-click on the file
And selects option "New scan project"
```

```
Then new project id is returned to the user
And project with empty configuration and default enabled check list is created into database

Scenario: Set existing scan project parameters (KR6 and KR7)
Given a generated IaC code from DOML
When user navigates to the IaC document/zip
And right-click on the file
And selects option "Config existing scan project"
Then scan project prompt configuration appears
And user enters previously generated project id
And user provides set of parameters (secrets, tokens, credentials) for individual tools
And user selects the list of enabled check tools
When user confirms configuration parameters
Then project configuration parameters are updated in database

Scenario: Initiate IaC Scan runner for pre-defined scan project configuration (KR6 and KR7)
Given A generated IaC code from DOML
When user navigates to the IaC document/zip
And right-click on the file
And selects "Scan for existing project"
Then prompt asking for project id appears
And user set response type (HTML or JSON)
[Optional] And user gives the list of preferred checks
Then IaC scan runner is invoked for intersection(enabled, [preferred]) checks
And a response is returned

Scenario: Scan tool response visualization (KR6 and KR7)
Given a generated IaC code from DOML
When user navigates to the IaC document/zip,
And right-click on the file
And selects option "View all scans"
Then list of all scan tasks appears
[Alternative 1]
And user selects one of them by doing left click
And selects result type (JSON or HTML)
Then scan result details are retrieved from database
And response file generated in project directory
And result is shown in window (mini-browser for HTML, text editor for JSON)
[Alternative 2]
And user selects one of them by doing right click
Then prompt for scan result deletion appears
And user confirms deletion of scan result
And scan result is removed from database

Scenario: Scan project management (KR6 and KR7)
Given a generated IaC code from DOML
When user navigates to the IaC document/zip,
And right-click on the file
And selects option "View all projects"
Then list of all projects by current user appears
[Alternative 1]
And user selects one of them by left-clicking
Then list of all scan results for given project appears
[Alternative 2]
And user selects one of them by right-clicking
Then prompt for scan project deletion appears
```

```
And user confirms deletion of scan project
And scan project is removed from database
```

### 2.3.1 Fitting into overall PIACERE Architecture

The results described in this deliverable, the ***IaC Security Inspector*** and ***IaC Component Security Inspector*** fit into the **Vulnerability tool** section of **PIACERE Design Tools** [2]. The services can be initiated by IDE after IaC will be generated from the DOML language. The **inputs** of the IaC Security Inspector and Component inspector are simplified be simplified to allow scanning IaC packages (such as zip or tar files). The **outputs** will be formatted as JSON and will be sorted by tools. The IaC generated from the DOML in inspected by IaC Security Inspector and IaC Component Security Inspector, depicted as KR6 and KR7 (IaC Scan Runner) in Figure 1.

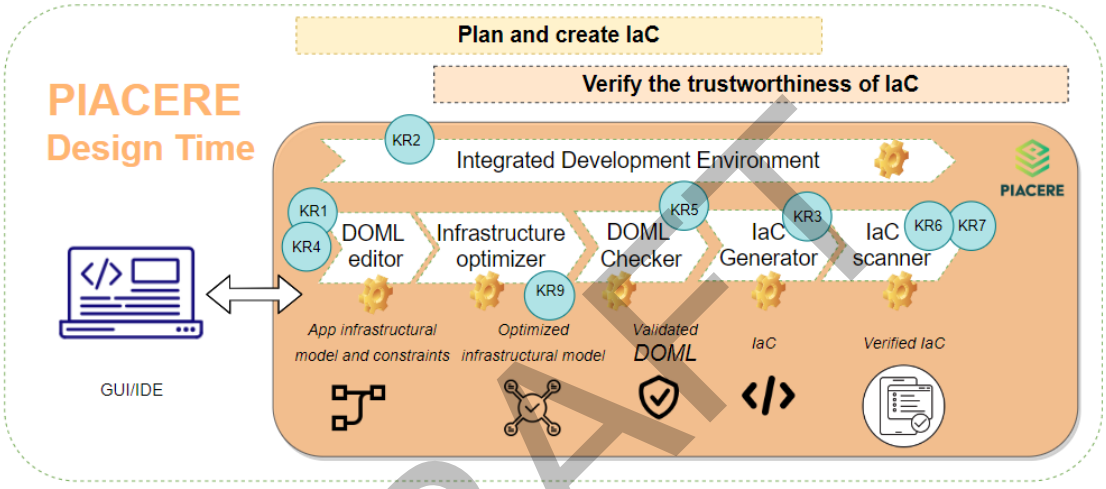


*Figure 1. PIACERE design time components*

Within the PIACERE framework**, only IDE currently access the IaC Scan Runner** through its API. Other components do not interact with it for now, but in the future could be useful to add the IaC Scan Runner also in the CI/CD jobs, to re-check the IaC just before continuous delivery step. The IaC from the design phase is unchanged, but there are two corner cases which we would catch with this additional step. First, it might be, that user would forget to run the IaC Scan runner in the design time, and in this case, we would catch potential issues. Secondly, the IaC components and IaC languages are improving trough the time, this means that even a perfect IaC today, might have many vulnerabilities or misconfigurations tomorrow or after one month. To maintain the IaC in a top-notch shape, it is necessary to schedule IaC scanning just before use, and continuously when the application is already in runtime. Many potential issues could be detected this way.

## 2.4 Technical description

The **IaC Security Inspector [KR6] and Component security Inspector [KR7]** are components of the software result called **IaC Scan Runner**. In this section we first in detail present the updates of KR6 and KR7, which focus on the respective fields. In the last subsection we present the updates that can be attributed to both KRs and are done directly on IaC Scan Runner. These updates are mainly covering **user experience, integrational aspects and internal processing** of individual scans and results.

### 2.4.1 IaC Security Inspector updates – Y2

The security inspector has gained much more checks after the D4.4 deliverable. Many of new checks were already available in Y1 version of the **IaC Scan Runner**. Old, new, and abandoned checks are listed in the Table 3. The planned list from D4.4 [1] has been extended for three checks – **cloc, Snyk and SonarCloud** – and we selected three to not be integrated for now. Together we now support 22 checks, which is 14 checks more than it was integrated in the report time of D4.4. The summary of current state related to available checks is given in Table 3.

*Table 3 List of old, new and abandoned checks of KR6 an KR7*

| IaC Check | Target IaC entity | Enabled (by default) | New from D4.4 |
|---|---|---|---|
| xOpera TOSCA parser | TOSCA | yes | yes |
| Tosca-parser | TOSCA-parser | No | abandoned |
| Ansible Lint | Ansible | yes | no |
| Steampunk Scanner | Ansible | no | yes |
| TFLint | Terraform | yes | no |
| tfsec | Terraform | yes | no |
| Terrascan | Terraform | yes | no |
| Terrafirma | Terraform | No | abandoned |
| yamllint | YAML | yes | no |
| Pylint | Python | yes | no |
| Bandit | Python | yes | no |
| PyUp | Python | No | abandoned |
| Safety | Python packages | yes | no |
| Gitleaks | Git repositories | yes | yes |
| git-secrets | Git repositories | yes | yes |
| Markdown lint | Markdown files | yes | yes |
| hadolint | Docker | yes | yes |
| Gixy | Nginx configuration | yes | yes |
| ShellCheck | Shell scripts | yes | yes |
| ESLint | JavaScript | yes | yes |
| TypeScript ESLint | TypeScript | yes | yes |
| HTMLHint | HTML | yes | yes |
| stylelint | CSS and other styles | yes | yes |
| Checkstyle | Java | yes | yes |
| cloc | Multiple components | yes | yes |
| Snyk | Multiple components | no | yes |
| SonarScanner | Multiple components | no | yes |

Reference on the documentation page (GitHub)[1].

### 2.4.2 Component Security Inspector updates Y2

In the second year of the project, we focused also on improving the performance of component scanners. From **PIACERE use-cases, customers and IaC practitioners** we get the **feedback** about the component **inspection of Ansible**. The Ansible was in recent years divided from one single component to a *core* component developed by core team and *collections,* that can be maintained by opensource community or any stakeholder. This means that whenever you write

[1] https://xlab-si.github.io/iac-scanner-docs/02-runner.html

an Ansible code you are relying to collections that **can vary in maturity**, **concepts** and **more** - means it is significantly **prone to errors**. The collections are maintained and upgraded during the time, which means, if we try to update the IaC code, we need to spot the differences between the old version and new version and try to figure out if we need to make any actions on our IaC code. In comparison to other language component checks **this one is the most critical**. Due to that drastic change in the Ansible architecture, the lack of component inspecting tools is more evident here than in others.

To tackle this matter a special team inside XLAB is **evolving IaC related tools** for enterprises and end customers. From the ideas of IaC Scan Runner and customers we **evolved XLAB's Steampunk Scanner to a Steampunk Spotter**. This tool is proprietary and will be available for Enterprise PIACERE users through IaC Scan Runner to improve the component scanning.

Steampunk Spotter ensures trustable automation by providing an assisted automation writing tool that analyzes and offers recommendations for different types of Ansible content (task files, playbooks, roles, collections). Currently Spotter targets scanning and analyzing Ansible content but will extend its support to other IaC tools, such as Terraform and Pulumi in the future. IaC Scan Runner uses Spotter CLI to interact with Spotter and to scan Ansible content.

The best thing about the Component Security Inspector using the Steampunk Spotter is the idea, that Spotter sometimes can update and automatically rewrite the IaC files. This means that when the DOML is translated to IaC by the ICG component, the resulting IaC could be easily upgraded by the IaC Scan Runner without changing the DOML or upgrading ICG. This idea is very promising and still under the investigation of how practically develop and provide the functionality to the users.

### 2.4.3 Extended scan workflow

The updated high-level workflow of **IaC Scan Runner** focusing on new features is depicted in Figure 2. In this version, as it can be seen, there are three main novelties: 1) compatibility determination step – ensuring that scan will be performed only for compatible files within IaC archive 2) re-use of user-defined configuration – that hold information about individual tool configuration and enabled checks 3) result persistence – storing the result summary, enabled thanks to introduction of persistence layer. However, in case that underlying database is unavailable, the tool will be used in stateless mode without storing the scan results and scan project configuration, as it was completely described in the previous document.

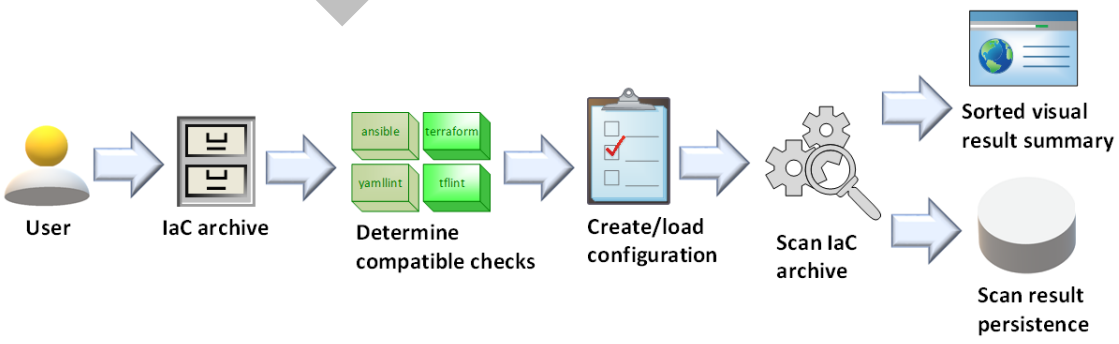


*Figure 2 High-level workflow of updated IaC and Component Security Inspectors*

Moreover, Figure 3 and Figure 5 show the use case and sequence diagram focusing on the extension with respect to previous version showing the new capabilities enabled by **persistence layer** in more details, respectively. When it comes to user perspective, in persistence-enabled mode, it is now possible to create scan projects and **assign individual scan tasks to projects**.

Apart from individual tool configurations parameters, projects also persist the information about the enabled checks, so user does not have to enable/disable specific checks each time similar IaC archive is scanned. Additionally, basic capabilities related to scan result and project browsing and management are added to the API. Finally, it is also possible **to enable or disable periodic job** that will clean scan results older than 14 days.

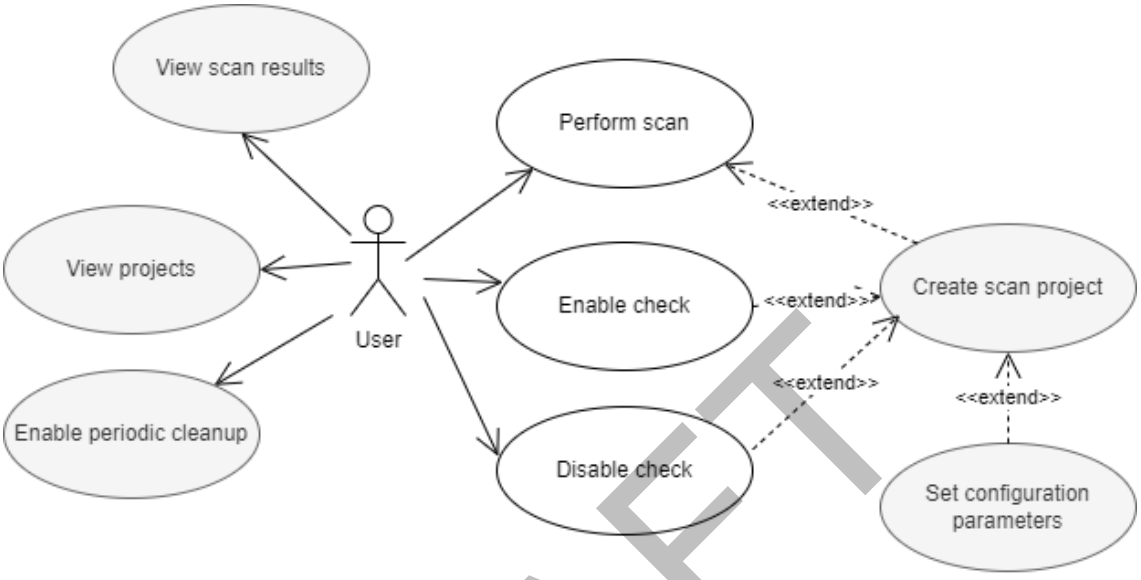


*Figure 3 Use-case diagram showing new features (gray) related to project configuration and result persistence of IaC and Component Security Inspectors*



In the first step of the sequence, it is possible to create a new scan project. As outcome, project id is returned to the user/IDE. After that, user/IDE is able to update the list of checks that will be executed against IaC archive by enabling or disabling individual checks of the scan workflow. Additionally, there might be some tools which require **client-specific or configuration details,** such as tokens, secrets, passwords etc. In order to ensure extendibility with such scan workflow tools in future, we also give the ability to the **user/IDE to provide** such parameters and **persist** them for **later re-use**, along with list of enabled checks. In such scenario, such checks are removed from the list of enabled checks if additional parameters are not provided. Finally, once the project is created and configuration finished, the scan of IaC archie will be performed. The first new step of scanning process is preliminary IaC archive file type list construction, in order to find out which kinds of scripts are present within the archive. After that, the list of file types is used as input of **compatibility matrix** with aim to retrieve the list of actual checks that are compatible with the provided IaC archive. Such approach is used in order to reduce the time needed for execution of non-compatible checks that are irrelevant for the given archive type. Therefore, the previously defined list of selected and enabled checks are filtered in order to distinguish only the ones that are compatible with the archive. Furthermore, there are several possible scenarios for final check list construction, depending on presence of certain input parameters, as shown in Table 2.

On the other side, if tool is used in **non-persistent** mode or while projects are not used even in persistent mode, there is still update of the flow due to introduction of compatibility matrix mechanism. In that case, the initial list of enabled checks is the same as the one in the previous version of the tool. While there is an initial default list of enabled checks, users can further enable/disable them individually. Moreover, there are two possible situations when it comes to

the checks that will be executed, depending if the list of selected checks is provided or no. These two cases are summarized on the bottom of Table 4 .

*Table 4 Check list determination in various scenarios*

| Persistence | Selected checks | Enabled checks | Projectid | Final list |
|---|---|---|---|---|
| Yes | Non-empty | Non-empty | Provided | Selected ∩ Enabled (and Configured) ∩ Compatible |
| Yes | Empty | Non-empty | Provided | Enabled (and Configured) ∩ Compatible |
| Yes | Non-empty | Empty | Not provided | Selected ∩ Default ∩ Compatible |
| Yes | Empty | Empty | Not provided | Default ∩ Compatible |
| No | Non-empty | Default with additional enabled/disabled | - | Selected ∩ Enabled ∩ Compatible |
| No | Empty | Default with additional enabled/disabled | - | Enabled ∩ Compatible |



After that, once the final list of checks to be executed is known, IaC Scan Runner provides the archive to target scan workflows and collects the returned logs. Moreover, the returned logs for various checks are processed in order to create a scan result summary. In this case, for each distinct scan workflow tool, specific string or pattern is searched within the returned log in order to determine whether the check passed or failed. Otherwise, if check was enabled and configured, but not found within the list of compatible checks, then its status will be recorded as “No files” within the results summary. Two types of summaries can be returned to the user/IDE, depending on the selected response type: 1) JSON – raw JSON file, the same form is persisted within the database 2) HTML page – tabular, overview of scan results containing various visual annotations (colours, sorting) to ease the result insight from the perspective of end-users, together with other useful information about the executed scan.

Furthermore, later, it is possible to browse the projects and corresponding scans. User/IDE can see all their projects and select the one that they want to browse. After that, the list of individual scans performed within the same project. Moreover, user selects one among the scans, so additional details about the scan can be visualized.

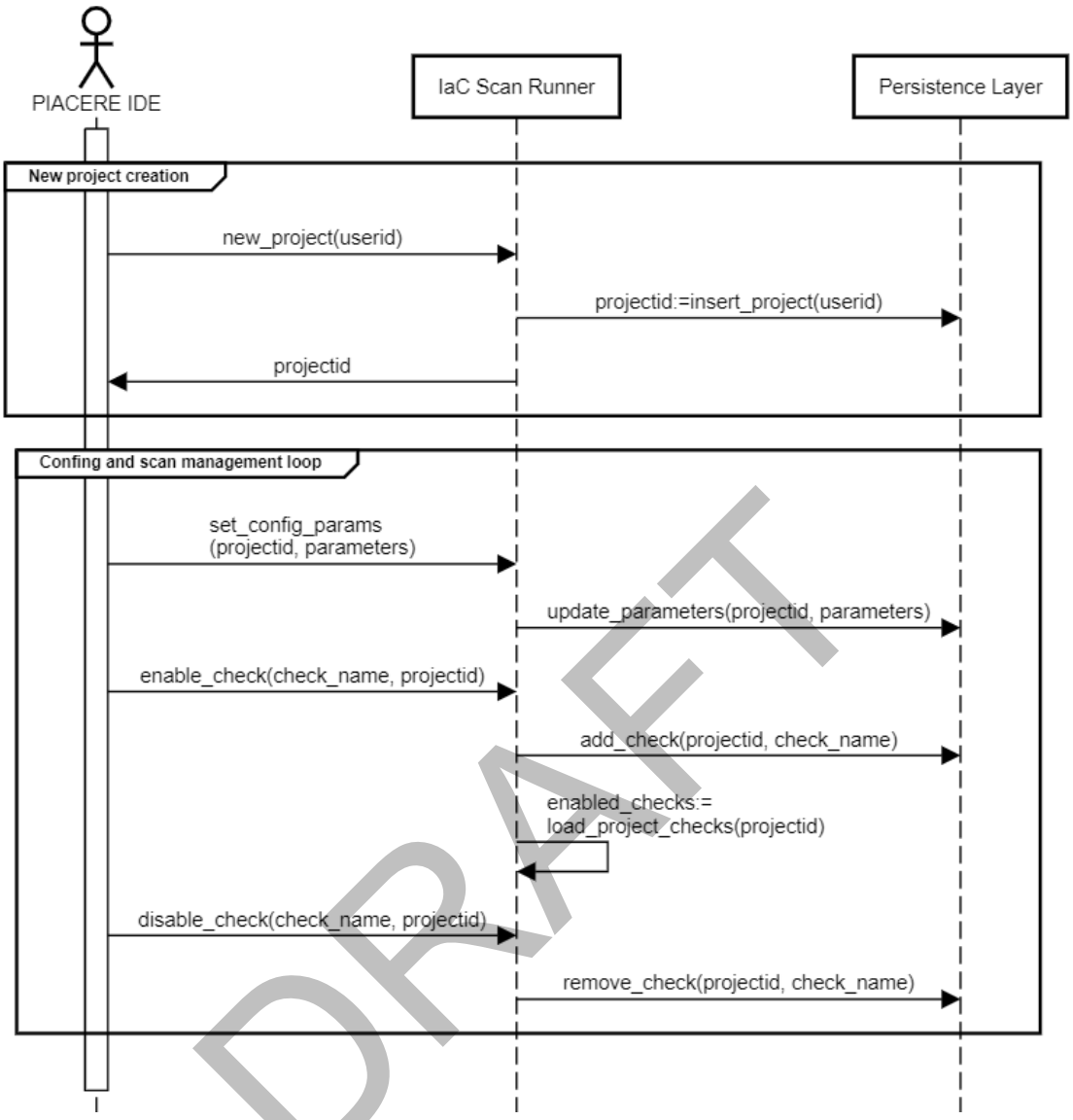


*Figure 4 IaC Scan Runner: project creation and setting the configuration*

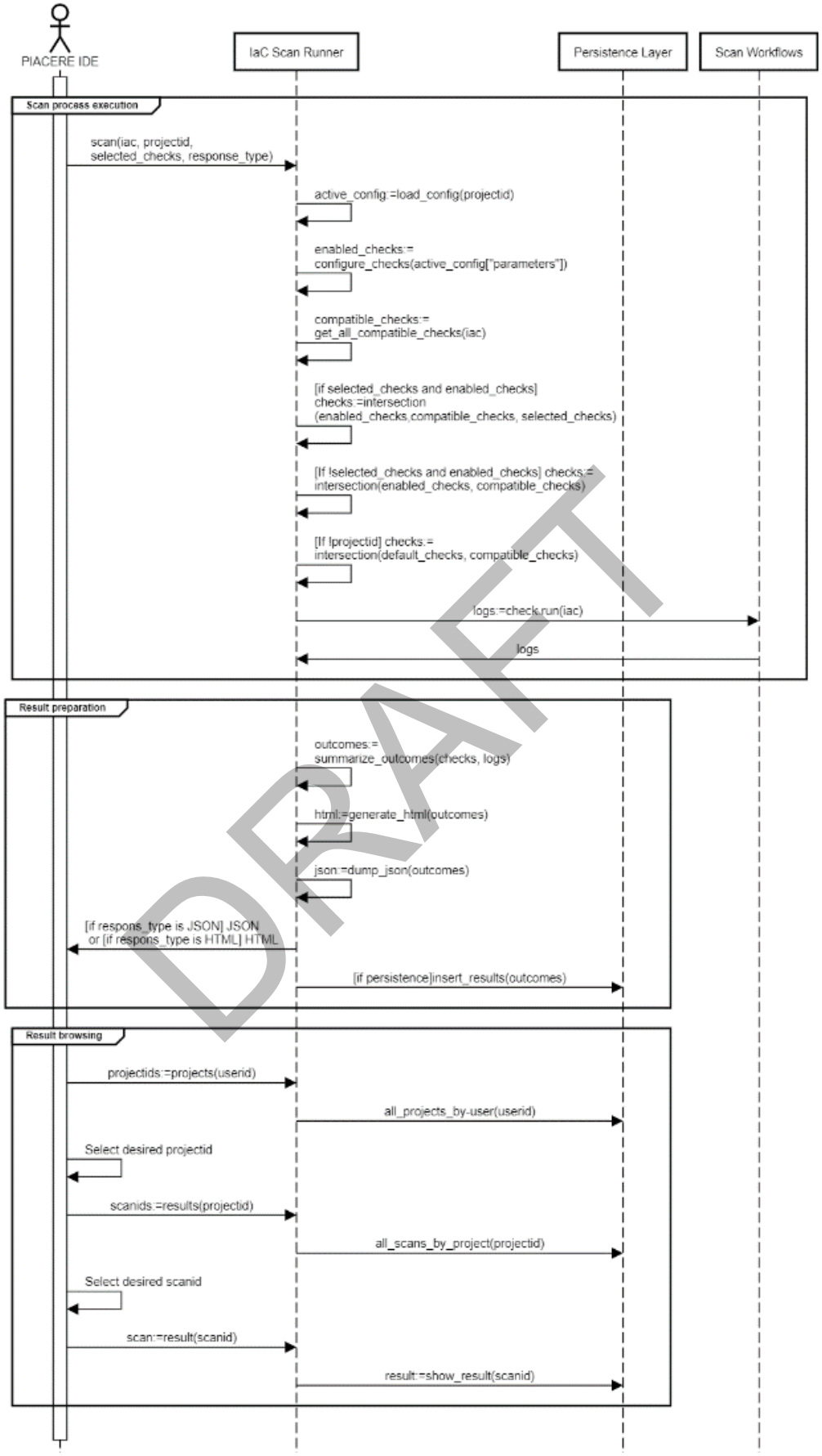




*Figure 5 Updated IaC Scan Runner scan workflow with result persistence and pre-defined configuration*

### 2.3.3. Scan projects and configuration

Another concept relevant to IaC archive scanning process is a scan project. It consists of one or more IaC scan runs belonging to a particular user and sharing the same settings and tool configuration.

**IaC Scan Runner** can be used in both **transient** and **persistence**-enabled **mode**. When it comes to transient mode, user-related data is not persisted, so scan projects and configurations are only relevant to persistent mode. Moreover, even in persistent mode, **user configuration management can be enabled or disabled** (only scan results would be stored into DB).

However, in case when it is enabled, it is necessary to provide project id for each scan run task. After that, the project id is used to load project-related info from the document store. First, the list of enabled checks is loaded and compared against the list of returned compatible checks. Additionally, the assigned active configuration is loaded for setting up some of the check tools (if available). Moreover, the parameters are extracted from active configuration to set-up the parameters relevant to specific scan tools, such as secrets, tokens or licenses (if required by some of them which are not free). However, it is assumed that set of default free check tools is present even without additional configuration in open-source variant of IaC Scan Runner. Finally, the retrieved parameters are used for tool configuration, and scan is performed for the scan checks that are intersection of enabled, compatible and configured tools. The previously described scan project configuration management workflow is depicted in Figure 6.

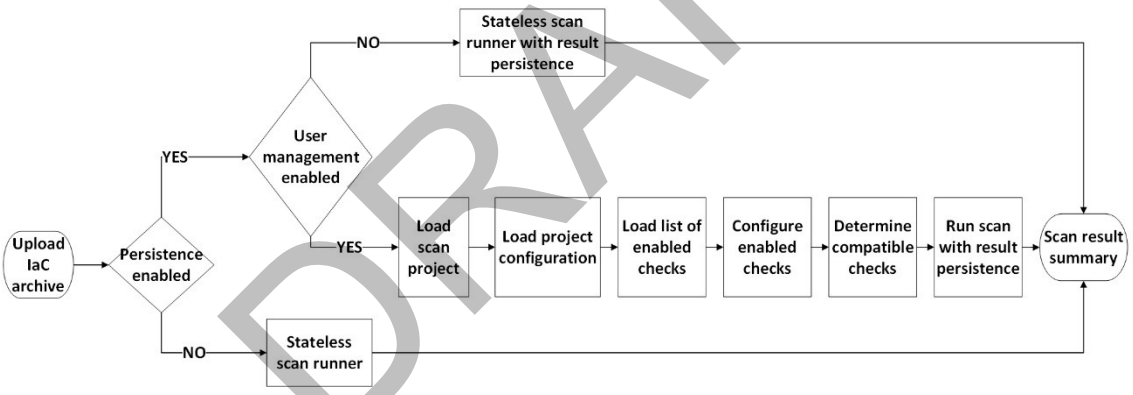


*Figure 6 . Illustration of scan project and configuration workflow*



### 2.4.4 Compatibility matrix

*Another novel concept introduced to IaC Scan Runner tool in this version is **compatibility matrix**. Its purpose is to aid the analysis of the submitted IaC archive in pre-scanning phase in order to determine the **checks that are compatible** with the file types present inside the archive itself.*

Table 5 summarizes the current version of compatibility matrix, while it is still work in progress. The first column denotes the name of particular file type, while the second shows list of compatible scan tools which are suitable for that file type. As it can be seen, there are two special categories:

1) other – for all the file types that are not targeted by particular scanning tools
2) common – general checks that are performed against any IaC archive submitted

However, in future it is planned to further investigate compatible file types and which kind of scan tools can be leveraged in specific context. In case that the file does not belong to any of the categories from the left, then no scan will be performed. The purpose of such approach is to avoid execution of incompatible scan tools and possible reduce the time needed for scan process of the IaC archive.

*Table 5 Compatibility matrix overview*

| File type | Scan workflow |
|---|---|
| Terraform | tfsec, tflint, terrascan |
| Yaml | yamllint |
| Shell | shellcheck |
| Python | pylint, bandit, pyup-safety |
| Ansible | ansible-lint, steampunk-scanner |
| Java | checkstyle |
| JavaScript | ts-lint, es-lint |
| Html | htmlhint |
| Docker | hadolint |
| Common | git-leaks, git-secrets, cloc |

### 2.3.3. Scan output summary

This section describes the new features and updated format related to scan result summarization and their visualization, taking into account both the generated JSON and HTML web page which is shown to the end user.

#### *2.3.3.1 JSON*

In current version of the scanner tool, the information returned as output is extended and represents a summary of the individual checks performed against the user-provided IaC archive. For each of the scan runs executed, the following fields are contained

- **scan** - the name of the tool applied
- **status** - what is the outcome of scan:
  - ***passed*** – the check passes, no problems detected
  - ***problems*** - there is an issue detected by the particular scanning tool
  - ***info*** – results of scans that provide additional information about the IaC archive (like cloc, for example which provides file types and number of code lines per file), so they are neither “passed” or “problems”
  - ***no files*** – no scanning was performed, as there were no compatible files
- **files** - list of files scanned using that particular tool
- **log** - contents of particular scan tool log in raw textual format
- **verdict** – the final outcome of the scan, which has value “passed” only if there are no scans that returned “problems” as result. Otherwise, it takes value “problems”. For evaluation of the final verdict, the scan results with “info” outcome are not taken into account.

Apart from result summary fields denoting scan outcomes as previously described, the following additional fields are also included when scan record is persisted into database, containing additional auxiliary info:

- **time** – date/time in format "%m/%d/%Y, %H:%M:%S", representing the moment when the result summary of a scan was persisted into document store

- **uuid** – an identifier (uuid4 type [3]), randomly generated for each scan run, which is used in order to distinguish the submitted scan tasks, while the user of the tool itself is aware of this value, apart from Mongo DB's _id that is used only internally
- **archive** – the name of IaC archive containing files that were scanned
- **execution-duration** – time required for scanning the given archive for selected list of compatible scans, given in seconds
- **projectid** [optional] – identifier corresponding to the scan project with assigned enabled check list and configuration parameters. If not set or persistence disabled, the default check list will be used.

The form of the returned JSON output is illustrated in what follows:

```
{ "scan_tool1": { "log": "log_text_tool1", "files": ["file1"…"filen"],
"status": "Passed"},

...

{" scan_toolM": { "log": "log_text_toolK", "files": ["file1"…"filej"],
"status": "Passed"},

…

{" scan_toolM": { "log": "log_text_toolM", "files": ["file1"…"filek"],
"status": "Info"},

"uuid": "uuid4",

"archive": "archive_name.zip"

"time":  "%m/%d/%Y, %H:%M:%S",

"execution-duration": "TIME_SEC"

"verdict": "Passed"

"projectid": "project1"

}
```

### 2.3.3.2 HTML summary

Beside presented JSON option to retrieve results, it is also possible to retrieve the output of a scan process in form of HTML page. It is generated based on the previously described JSON file and includes the mentioned fields, represented in tabular form, as shown in Figure 7.

**Scan results**

| Aspect | Measure |
|---|---|
| Archive name | terra_example.zip |
| Run on | 10/18/2022, 22:45:50 |
| Time spent (seconds) | 0.314 |
| Final verdict | Problems |

| Scan | Status | Files | Log |
|---|---|---|---|
| tfsec | Problems | ['main.tf', 'variable.tf'] | Result 1 [general-secrets-sensitive-in-variable][CRITICAL] Variable 'variable.password' includes a potentially sensitive default value. /iac-scan-runner/src/171b7a/terra_example/variable.tf:8 5 \| 6 \| variable "password" { 7 \| type = string 8 \| default = "pa$$w0rd" string: "pa$$w0rd" 9 \| } 10 \| 11 \| variable "vc_username" { Legacy ID: GEN001 Impact: Default values could be exposing sensitive data Resolution: Don't include sensitive data in variable defaults More Info: - https://tfsec.dev/docs/general/secrets/sensitive-in-variable#general/secrets - https://www.terraform.io /docs/state/sensitive-data.html times ---------- disk i/o 1.119607ms parsing HCL 9.975µs evaluating values 529.362µs running checks 797.751µs counts ---------- files loaded 2 blocks 13 evaluated blocks 13 modules 0 module blocks 0 ignored checks 0 results ---------- critical 1 high 0 medium 0 low 0 1 potential problems detected. |
| terrascan | Problems | ['main.tf', 'variable.tf'] | /bin/sh: 1: /iac-scan-runner/tools/terrascan: not found |
| git-secrets | Problems | ['main.tf', '.gitkeep', 'variable.tf'] | fatal: not a git repository (or any of the parent directories): .git |
| tflint | Passed | ['main.tf', 'variable.tf'] | |
| git-leaks | Passed | ['main.tf', '.gitkeep', 'variable.tf'] | time="2022-10-18T22:45:50Z" level=info msg="scan time: 96 microseconds" time="2022-10-18T22:45:50Z" level=info msg="commits scanned: 0" time="2022-10-18T22:45:50Z" level=info msg="No leaks found" |
| cloc | Info | ['main.tf', '.gitkeep', 'variable.tf'] | 2 text files. classified 2 files 2 unique files. 0 files ignored. github.com/AlDanial/cloc v 1.92 T=0.01 s (135.6 files/s, 12271.2 lines/s) ---------- Language files blank comment code ---------- HCL 2 33 5 143 ---------- SUM: 2 33 5 143 ---------- |
| opera-tosca-parser | No files | | |
| ansible-lint | No files | | |
| yamllint | No files | | |

*Figure 7 HTML page visualizing scan summary*

As it can be seen, the page consists of the two main parts: auxiliary info and scan results. The first table provides the following information to the user:

- **archive name** – what is the name of the scanned IaC zip archive
- **run on** – the timestamp of the moment when scanning of the provided archive was performed
- **time spent (**seconds) - processing time which was spent for IaC archive scanning for the selected checks that are compatible
- **final verdict** – the overall outcome of the check, which has value “Passed” only in case that no problems were reported by any of the scanning tools.

The second part depicts scan result summarization and additionally, it includes the aspects of scan outcome prioritization, considering the actual status of the performed scans and leveraging them in context of visualization. Therefore, the scan results will be grouped according to the outcome and coloured the following way:

- Problems – red colour, shown first
- Passed – green colour, second
- Info – yellow colour, third
- No files – greyed out, fourth

## 2.4.5 Prototype architecture

**IaC Scan Runner** is invoked via **PIACERE IDE** and interacts with it **via REST API**. Figure 8 provides high-level overview describing the interaction of IaC Scan Runner Components. Compared to previous version of the document, the main novelty are **Scan Database** and **Project Database** components (orange-coloured blocks). The first one encapsulates the capabilities related to scan result storage, management, and retrieval. Additionally, it is possible to **organize scan** tasks into projects as well, for what is responsible the second one. Moreover, we can assign a scan

configuration to a project, which can be later re-used for further scans within that project and avoid the repeating manual steps.

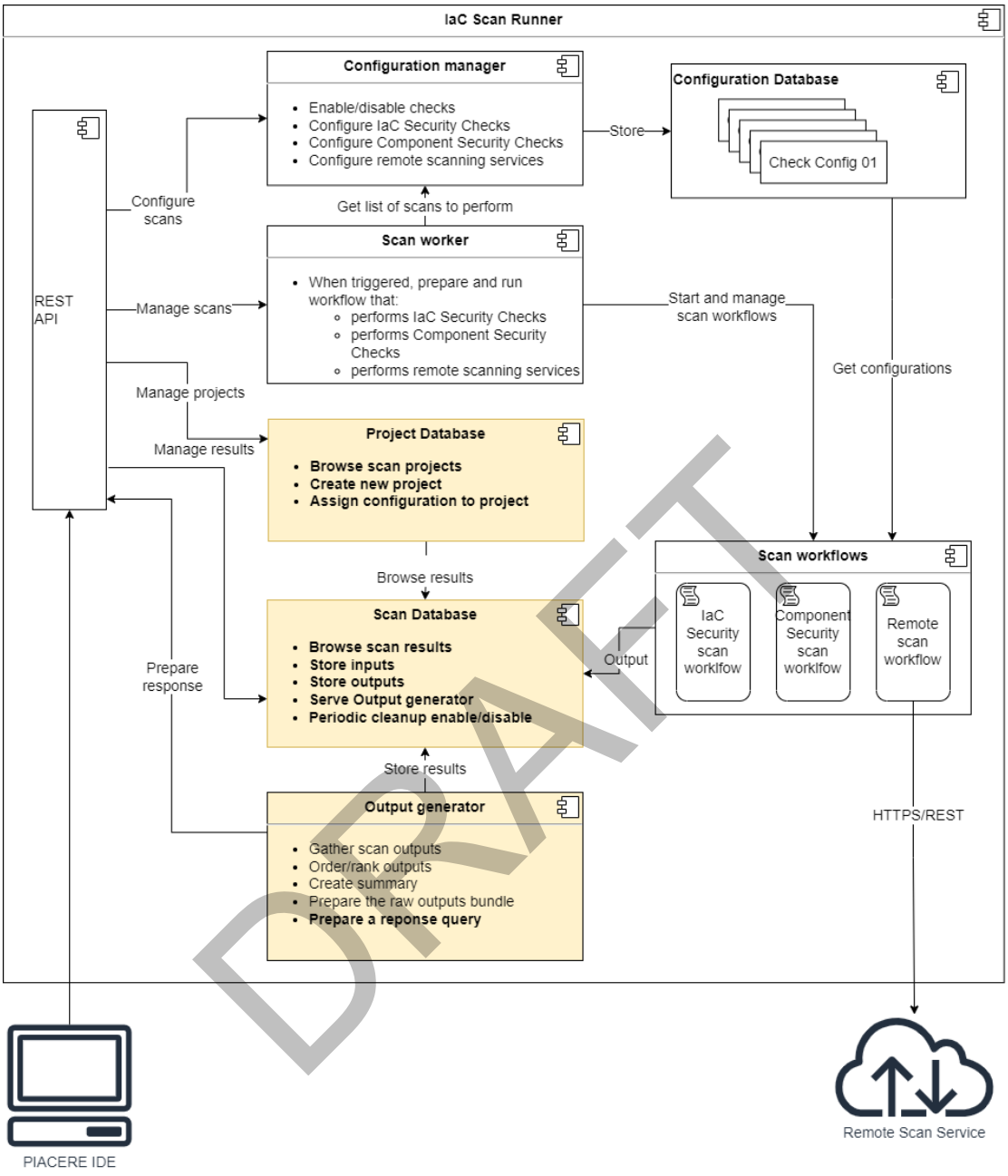


*Figure 8 IaC Scan Runner component schema*

On the other side, Figure 9 and Table 6 gives more detailed insight into main components of IaC Scan Runner on lower level that are leveraged by PIACERE IDE during the user interaction.

Each time new scan is performed by right-clicking on IaC archive, a scan request is sent to the ***ScanRunner*** component, forwarding the archive name, project id and list of preferred scans. After that, the corresponding scan project and its configuration are loaded, so the list of executed checks can be determined. Finally, IaC archive is scanned by each of these checks and retrieved results stored into database thanks to ***ResultsPersistence*** component.

Additionally, it is possible to create new projects within IDE relying on ***ScanProject*** component once the IaC archive is loaded. On the other side, there is also functionality to list all the projects by right clicking on IaC archive inside IDE. Furthermore, it is possible to browse the scan results by projects once they are returned.

Finally, IaC Scan Runner also provides the ability to persist scan project preferences which can be later re-used. For that purpose, ***ProjectConfig*** component enables management of both the list of enabled scan workflow checks and tool-specific parameters that are used for their proper configuration.

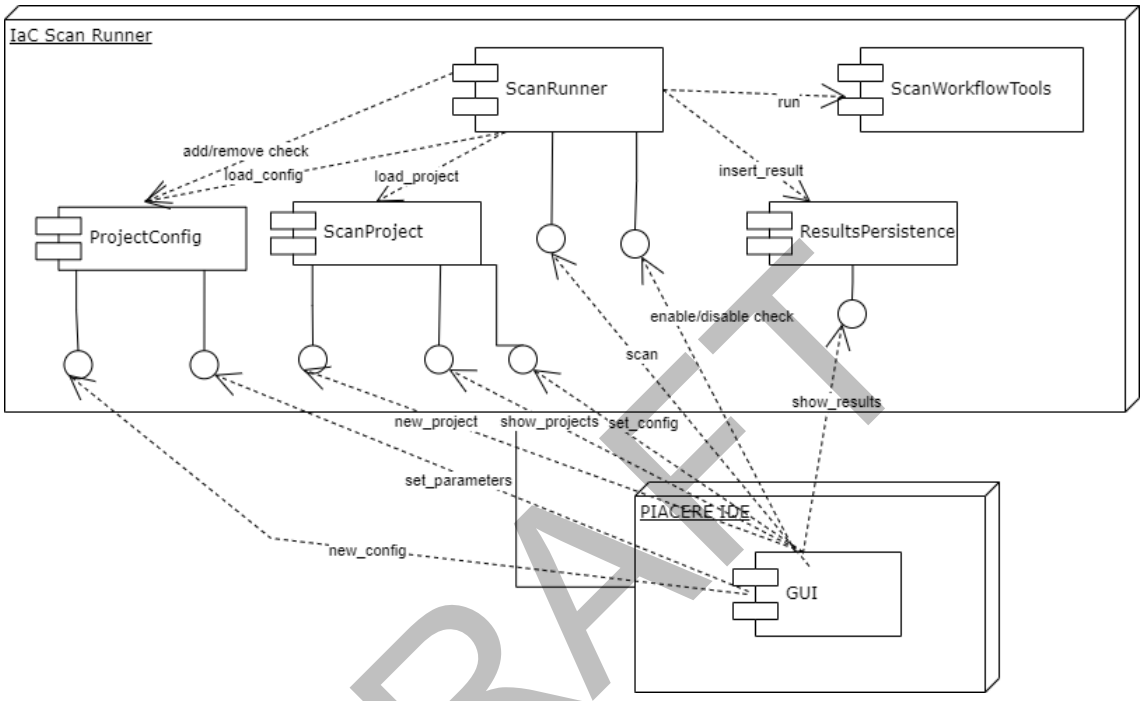


*Figure 9 IaC Scan Runner prototype components diagram*

### 2.4.6 Components description

Table 6 gives an overview of most important components together with their description.

*Table 6 Details of IaC Scan Runner component actions*

| Component | Description |
|---|---|
| ScanRunner | The main component that performs the whole scan workflow for provided IaC archive |
| ResultsPersistence | Data layer component responsible for management of scan result records – their insertion, deletion and retrieval. |
| ScanProject | Data layer component enabling the creation of new scan projects, assigning corresponding configuration to them and loading their configuration and preferences before scan workflow execution. |
| ProjectConfig | Data layer component for managing the creation of scan project configuration, which make possible the re-use of previously created user preferences. It gives ability to persist the list of enabled scan workflow checks and corresponding tool-specific parameters. |

| | |
|---|---|
| ScanWorkflowTools | A collection of individual external scan tools that can be invoked by ScanRunner and represent the part of its IaC archive scan workflow. |

### 2.4.7 Technical specifications

The new version of IaC Scan Runner includes addition of persistence layer for purpose of scan result storage and project configuration re-use. In our case, we make use of **MongoDB** [4], which is a document-oriented NoSQL cross-platform database. In our case, MongoDB Community is used, as it is open-source variant. It relies on **JSON**-alike documents with optional schemas. MongoDB stores data in form of **JavaScript** objects, referred to as documents. Official MongoDB database drivers are available for wide range of language, while Python is among them. Unlike relational SQL tables, MongoDB does not impose structural limits – data schemas are not enforced, so practically anything can be stored there. The main concepts of MongoDB are given as follows:

1) **document** – individual object which is persisted, analogous to row in relational database table
2) **field** – a single item of data within the considered document, similar to relational database column
3) **collection** – a set of documents similar to each other, analogous to SQL tables
4) **database** – a set of related collections, analogous to SQL database
5) **schema** – data structure definition, which is not obligatory in case of MongoDB, but still can be leveraged for validation purposes.

We decide to use **MongoDB** due to its flexibility when it comes to schema modification, as it will ease extendibility with new features and enhancement of output results with additional useful information in future. On the other side, IaC Scan Runner itself produces JSON results as outputs, which is compatible with Mongo.

In case of IaC Scan Runner, **pymongo** [5] library is used to persist and retrieve data from MongoDB document store from Python source code of IaC Scan Runner. It represents a high-level encapsulation of MongoDB basic functionality, such as CRUD operations.

However, the MongoDB instance itself is recommended to be run in an isolated environment, in form of a Docker container running distinct service which interacts with IaC Scan Runner.

# 3 Delivery and usage

## 3.1 Package information

The packaging information is the same as in deliverable D4.4, with very minor changes. For the sake of completeness, we are providing the same text here.

The **IaC Scan Runner** module is delivered as a Docker application including a service accessible through an API. The ***xscanner/runner*** [6] Docker image (Figure 10) is updated and published regularly on Docker Hub. The CLI that is currently able to run the API is available as *iac-scan-runner* Python package and is published on **PyPI** [7]. Both, API and CLI use semantic versioning for new releases and the latest available version is 0.1.9.

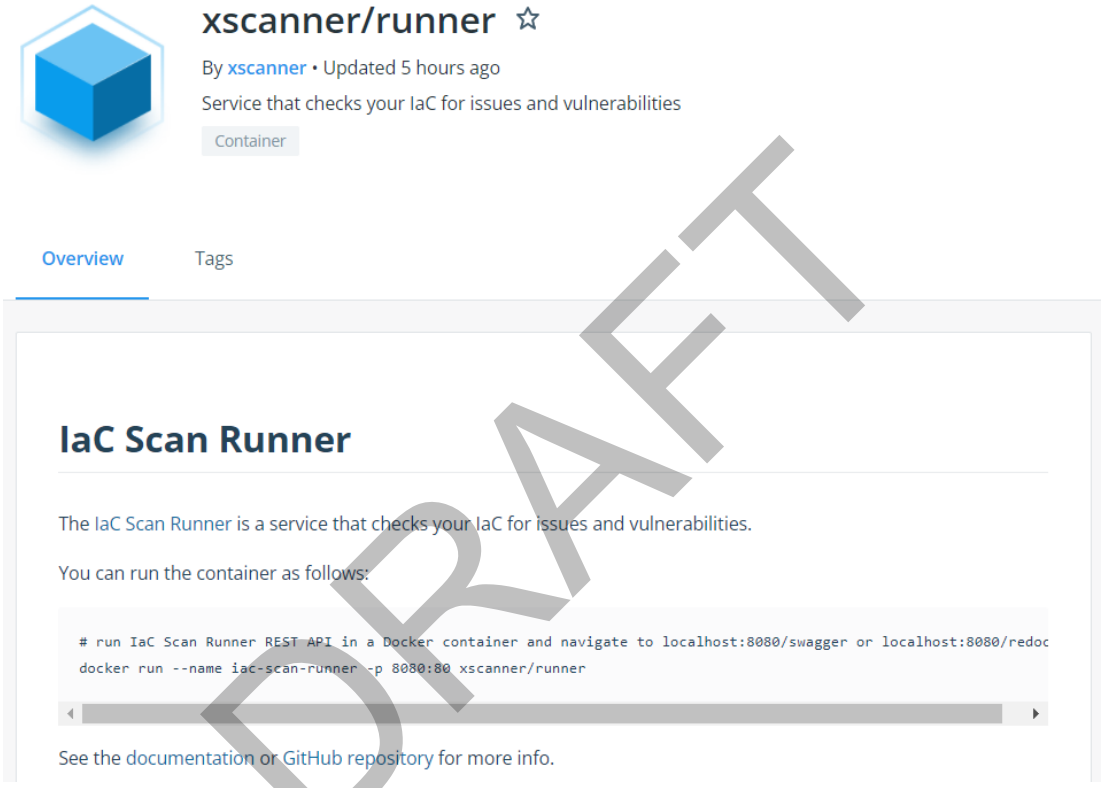




*Figure 10 Docker Hub repository for IaC Scan Runner*

Some of the IaC Scan Runner services are already available to the PIACERE consortium partners on public links:

- REST API: https://scanner.xopera.piacere.esilab.org/iac-scan-runner/
- Swagger UI: https://scanner.xopera.piacere.esilab.org/iac-scan-runner/swagger/
- ReDoc: https://scanner.xopera.piacere.esilab.org/iac-scan-runner/redoc/
- Documentation: https://scanner.xopera.piacere.esilab.org/docs/

The IaC Scanner REST API is protected by HTTP Basic Auth, to avoid the use from boots and robots.

A screenshot of GitHub Pages documentation for IaC Scan Runner from https://xlab-si.github.io/iac-scanner-docs/02-runner.html is given in Figure 11.

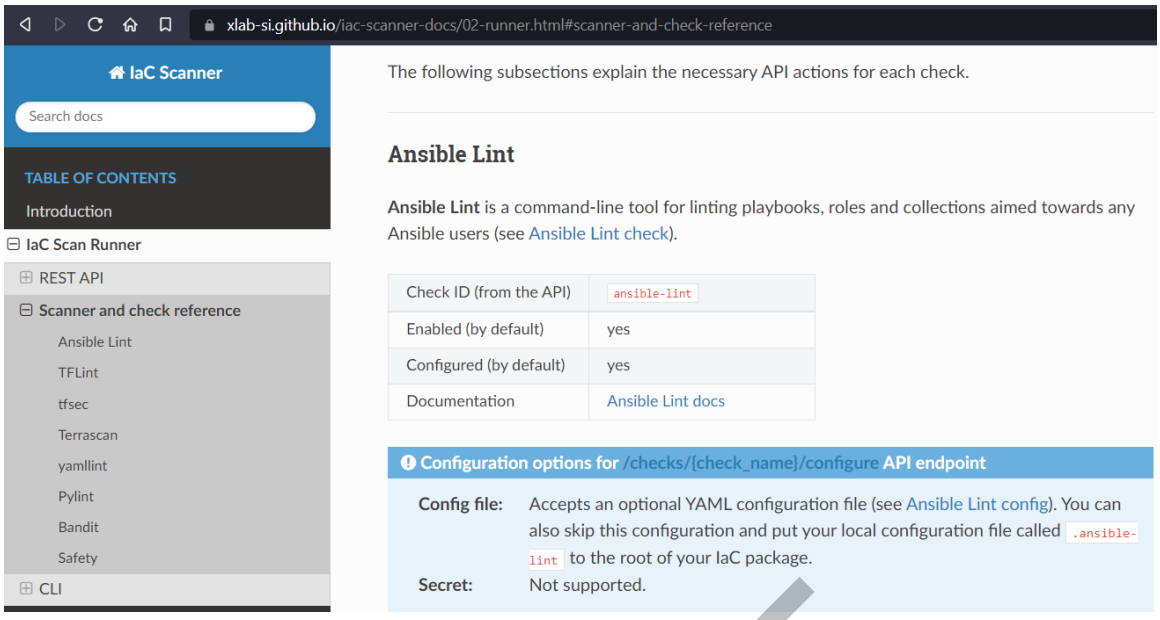


*Figure 11 IaC Scanner documentation page hosted on GitHub Pages*

## 3.2 Installation instructions

Compared to previous version, of the main difference is a requirement to run Mongo DB document instance for purpose of scan result persistence. In our case, we deploy it as Docker container. Therefore, the machine running IaC Scan Runner should have Docker Engine installed.

In order to ensure the creation of running Mongo DB instance for result persistence, it is necessary to run Docker Compose [8] script inside the root of the project with the following command

```
docker compose up -d
```

which will spawn two containers: IaC Scan Runner and Mongo DB (named “mongoservice”). The database is accessed externally via port 27017, while Scan Runner uses 8080, like shown below in Figure 12.

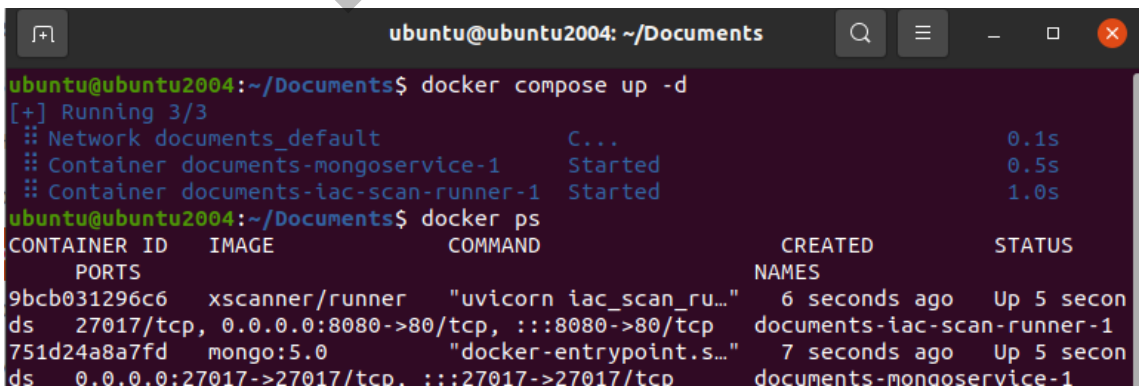


*Figure 12 Starting IaC Scan Runner and MongoDB persistence layer with Docker Compose*

However, it is still possible to use IaC Scan Runner without Mongo DB, so the scan results and project configuration will not be persisted into document store (but still dumped and returned as HTML pages or JSON to the user). For that purpose, it is necessary to check the values of the following environment variables, as shown in Table 7.

*Table 7 . Environment variables for persistence settings enable/disable*

| Variable | Meaning |
|---|---|
| MONGODB_CONNECTION_STRING | The connection string used by pymongo in order to connect do a database running inside container. The default value for enabling persistence with respect to our Docker Compose file would be:<br><br>mongodb://mongoservice:27017/ |
| SCAN_PERSISTENCE | Enables or disables using persistence at all. There are two possible values:<br><br>“enable” - if we want to use database for scan result persistence<br><br>“disable” - scan results will not be persisted<br><br>However, even enabling the persistence with improper connection string will result turning the persistence capabilities off |
| USER_MANAGEMENT | Enables or disables all the capabilities related to scan project and configuration management assigned to specific tool users |



Moreover, internal tool folders for output generation will be created after running `install-checks.sh` as well. Here, we are able to detect three directories:

1) **json_dumps** – holding the summaries of scan results
2) **generated_html** – web page reports generated based on JSON result summaries
3) **logs** – textual reports created by individual tools. Furthermore, this directory holds distinct folder for each of the scans that were run, while individual logs generated as result of scan workflow execution are contained inside.

After creation of these folders as one of outcomes created by running install-checks.sh, IaC Scan Runner is ready to be used.

## 3.3 User Manual

The IaC Scan Runner offers REST API, which can be tested/used via Swagger UI [9] that provides GUI, as shown in screenshot from Figure 13. Once the Docker Compose is up and iac-scan-runner service running, it is accessible via `locahost:8080/swagger`.

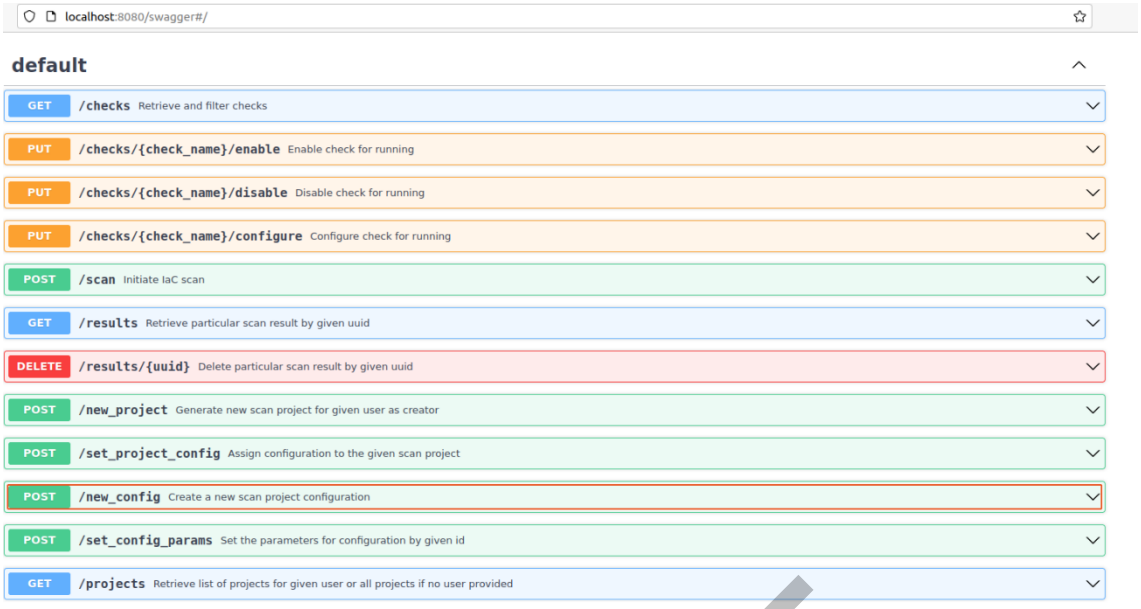


*Figure 13 Swagger UI for visual access to IaC Scan Runner REST API*

The most important method related to IaC archive is `scan`. Its input form is given as screenshot in Figure 14. As it can be seen, it is now possible to also provide **identifier of a scan project**, so the assigned scan configuration can be re-used. Additionally, it is also possible to proceed without **project id**, so the tool will consider the default configurations and scan tools enabled. After that, it is also possible to select whether the returned outcome would be JSON format or HTML summary (JSON is default). On the other side, there is also the possibility to specify the preferred checks that will be executed against IaC if they are compatible with the provided file types and properly configured. They are added as list of distinct strings within the checks section of the form. Furthermore, user has to specify the desired IaC archive from the disk which is about to be scanned. Finally, the scan process is started by clicking the "Execute" button of the form. Once the scanning of the provided IaC archive is finished, the result is shown below within Response body field, as displayed within Figure 15. It is possible to download the generated result in JSON form by clicking on Download button.

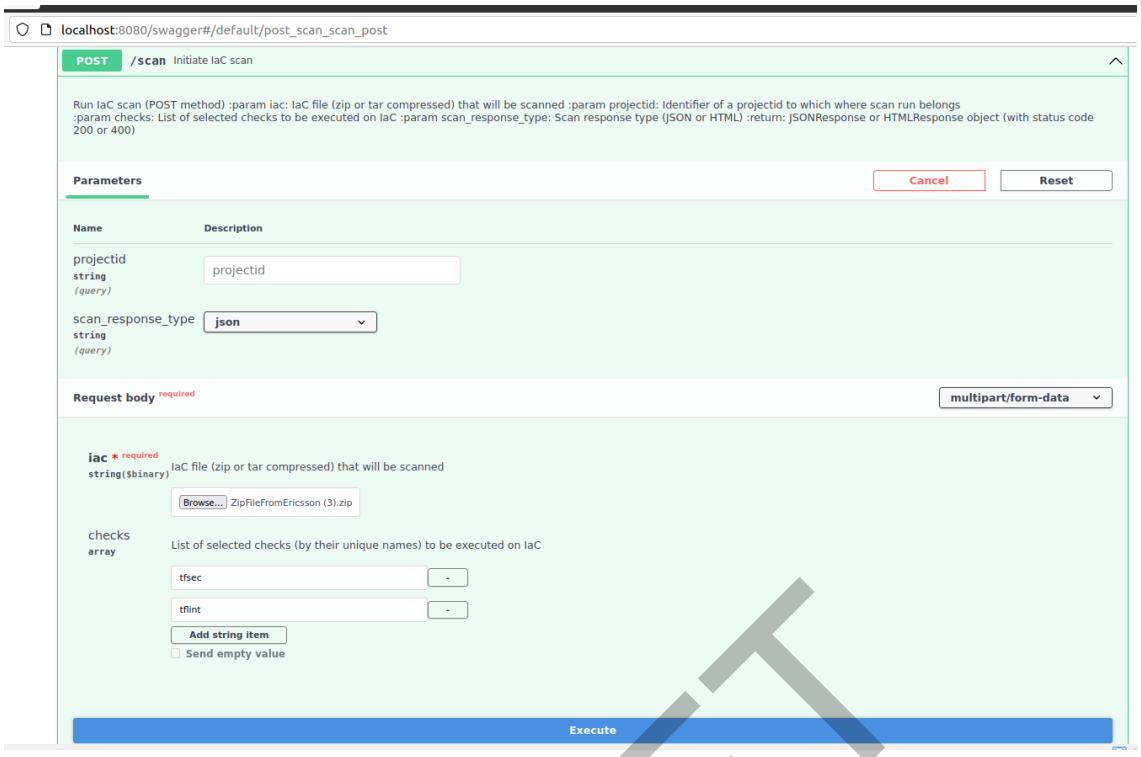


*Figure 14 Swagger UI IaC archive scan interface*

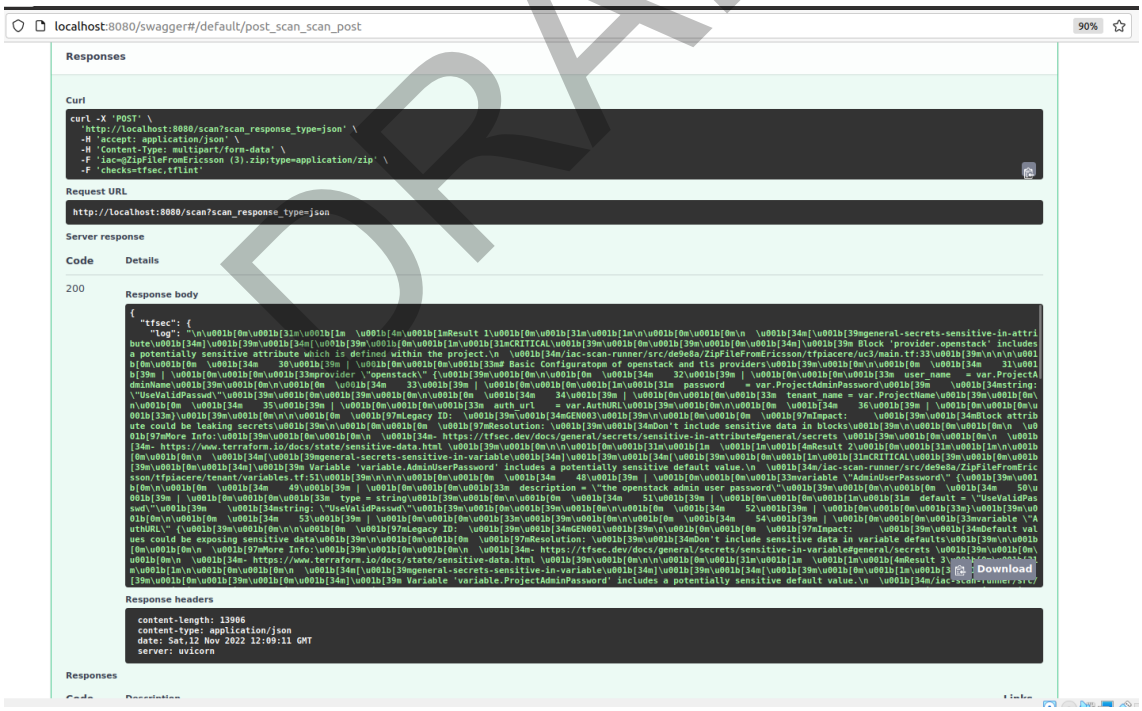


*Figure 15 UI IaC archive scan results*

In what follows, the currently available REST API will be described from two perspectives: scan results management and scan projects together with configuration aspects.

### 3.3.1 Scan result persistence

For each scan run triggered by user providing an IaC archive and selecting the list of desired scans, the aggregated results are stored into Mongo DB [] document store.

However, the persisted scans can be periodically cleaned-up by activating clean_old_scans service. Every day at 00:00, this periodic task is scheduled to check the age of all the scan outcome results currently residing within the document store and remove all of them that are older than 14 days.

In Table 8 , an overview of API which provides access to capabilities related to scan result persistence is given.

*Table 8 Scan results persistence API*

| API call | Parameters | Method type | Description |
|---|---|---|---|
| `/results/{uuid}` | uuid – A valid uuid4 value corresponding to a scan record persisted in database | GET | Returns the scan result from database with the provided uuid value as identifier.<br><br>Otherwise, returns all the persisted scan task information from the document store if uuid is not provided. |
| `/results/{uuid}` | | DELETE | Delete the scan result from database with matching uuid value. |
| `/scan_cleanup/{enable}` | enable – acts as enabler in case of value “1”, while disables the cron job if “0” is provided | PUT | Activates or deactivates cron job that will daily check the age of the existing scans in the database and clean each of them older than 14 days. |

### 3.3.2. Scan projects and configuration

Table 9 gives an overview of the API calls which are relevant for scan projects and configuration management-related features.

*Table 9 Scan project and configuration API overview*

| API call | Parameters | Method type | Description |
|---|---|---|---|
| `/new_project` | Creatorid – identifier of a user who created the scan project | POST | Creates a new scan project with randomly assgned uuid4 identifier for given identifier of creator. |

| | | | |
|---|---|---|---|
| /projects | | GET | Returns all the projects by particular user. If creatorid parameter is not provided, then all the scan projects will be returned. |
| /set_project_config | Projectid – identifier of a project whose config we want to modify<br>Configid – identifier of config which we want to assign to the desired scan project | POST | Changes the currently active configuration for the desired project. |
| /new_config | Creatorid – identifier of a user who created the configuration for a scan project | POST | Creates a new scan project configuration. |
| /set_config_params | Configid – identifier of config which we want to update with new parameter values<br>Parameters – a string which represents dictionary of check tool-related parameters, such as access tokens or secrets | POST | Assigns the set of check tool parameters to the desired scan project configuration. |



### 3.3.3. Extending the scan workflow with new check tools

At certain point, it might be required to include new check tools within the scan workflow, with aim to provide wider coverage of IaC standards and project types. Therefore, in this subsection, a sequence of required steps for that purpose is identified and described. However, the steps have to be performed manually as it will be described, but it is planned to automatize this procedure in future via API and provide user-friendly interface that will aid the user while importing new tools that will become part of the available catalogue that makes the scan workflow. Figure 16 depicts the required steps which have to be taken in order to extend the scan workflow with a new tool.

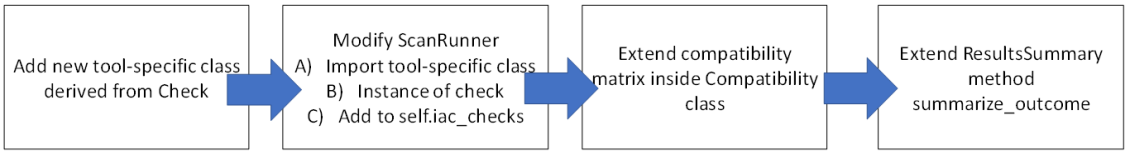


*Figure 16 Scan workflow extension steps*

#### *3.3.3.1. Step 1 – Adding tool-specific class to checks directory*

First, it is required to add a new tool-specific Python class to the checks directory inside IaC Scan Runner's source code:

```
iac-scan-runner/src/iac_scan_runner/checks/new_tool.py
```

The class of a new tool inherits the existing *Check* class, which provides generalization of scan workflow tools. Moreover, it is necessary to provide implementation of the following methods:

```
   1) def configure(self, config_filename: Optional[str],
secret: Optional[SecretStr])

   2) def run(self, directory: str)
```

While the first one aims to provide the necessary tool-specific parameters in order to set it up (such as passwords, client ids and tokens), another one specifies how the tool itself is invoked via API or CLI and its raw output returned.

#### 3.3.3.2. Step 2 – Adding the check tool class instance within ScanRunner constructor

Once the new class derived from Check is added to the IaC Scan Runner's source code, it is also required to modify the source code of its main class, called *ScanRunner*. When it comes to modifications of this class, it is required first to import the tool-specific class, create a new check tool-specific class instance and adding it to the dictionary of IaC checks inside `def init_checks(self)`.

A. Importing the check tool class

```
from iac_scan_runner.checks.tfsec import TfsecCheck
```

B. Creating new instance of check tool object inside init_checks

```
"""Initiate predefined check objects"""

        new_tool = NewToolCheck()
```

C. Adding it to self.iac_checks dictionary inside init_checks

```
self.iac_checks = {

    new_tool.name: new_tool,

    …

}
```

#### 3.3.3.3. Step 3 – Adding the check tool to the compatibility matrix inside Compatibility class

On the other side, inside file `src/iac_scan_runner/compatibility.py`, the dictionary which represents compatibility matrix should be extended as well. There are two possible cases: a) new file type should be added as a key, together with list of relevant tools as value b) new tool should be added to the compatibility list for the existing file type.

```
    compatibility_matrix = {

        "new_type": ["new_tool_1", "new_tool_2"],

        …

        "old_typeK": ["tool_1", …  "tool_N", "new_tool_3"]
```

```
}
```

#### *3.3.3.4. Step – Providing the support for result summarization*

Finally, the last step in sequence of required modifications for scan workflow extension is to modify class *ResultsSummary* (`src/iac_scan_runner/results_summary.py`). Precisely, it is required to append a part of the code to its method `summarize_outcome` that will look for specific strings which are tool-specific and can be used to identify whether the check passed or failed. Inside the loop that traverses the compatible checks, for each new tool the following structure of if-else should be included:

```
if check == "new_tool":
    if outcome.find("Check pass string") > -1:
        self.outcomes[check]["status"] = "Passed"
        return "Passed"
    else:
        self.outcomes[check]["status"] = "Problems"
        return "Problems"
```

## 3.4. Licensing information

IaC Scan Runner is licensed under open-source **Apache License 2.0**.

## 3.5. Download

IaC Scan Runner's current source code (regularly updated) is publicly available on GitHub within the following repository: https://github.com/xlab-si/iac-scan-runner and the documentation is available on GitHub pages: https://xlab-si.github.io/iac-scanner-docs/.

# 4. Conclusions

In the second iteration of deliverable *IaC Code security and component inspection* we present the progress of second year on the PIACERE IaC Security Inspector [KR6] and Component Security Inspector [KR7]. Both key results gained much attention, the KR6 plans of checks to be integrated was finalised and those are now integrated and operational in Y2 version. The component checker scans were also concluded, and we focused on filling the gaps of the use cases and Ansible users. With that we integrated the Steampunk Spotter in component check portfolio and also contributed with the improvement of that tool with the ideas gathered from the PIACERE project. The details of the process of choosing the checks was discussed in Section 2.1. Approach.

Another important progress was dedicated to the integrational and user experience of the KRs. The IaC Scan Runner that holds and manages both KRs has been improved by the result ranking and persistent features allowing the report management from result also after the scan has been successfully finished. This required a significant reconstruction of the application.

In the final phase we will finalise open tasks and focus on debugging, dissemination and exploitation of activities. Also worth to mention is that results of this deliverable were presented in scientific articles [10] [11].

# 5. References

[1] M. C. A. Luzar and G. Celozzi, "D4.4-IaC Code security and components security inspection-v1.0," 2021. [Online]. Available: https://www.piacere-project.eu/sites/d8piacere/files/Deliverables/D4.4-IaC%20Code%20security%20and%20components%20security%20inspection-v1_V1.0_20211130.pdf. [Accessed 21 November 2022].

[2] E. Morganti, A. Motta, L. Blasi, C. Nava and C. Bonferini, "D2.1 PIACERE DevSecOps Framework Requirements specification, architecture and integration strategy - v1," 2021.

[3] "UUID (Universal Unique Identifier)," [Online]. Available: https://www.techtarget.com/searchapparchitecture/definition/UUID-Universal-Unique-Identifier. [Accessed 20 November 2022].

[4] "MongoDB," [Online]. Available: https://www.mongodb.com/. [Accessed 21 November 2022].

[5] "PyMongo 4.3.3 Documentation," [Online]. Available: https://pymongo.readthedocs.io/en/stable/. [Accessed 21 November 2022].

[6] "IaC Scan Runner Docker image," [Online]. Available: https://hub.docker.com/r/xscanner/runner.

[7] "Python Package Index (PyPI)," [Online]. Available: https://pypi.org/project/iac-scan-runner/.

[8] "Docker Compose," [Online]. Available: https://docs.docker.com/compose/. [Accessed 20 November 2022].

[9] "Swagger UI," [Online]. Available: https://swagger.io/tools/swagger-ui/. [Accessed 20 November 2022].

[10] J. Alonso, P. Radoslaw and C. Matija, "Embracing IaC through the DevSecOps philosophy: Concepts, challenges, and a reference framework," *IEEE Software,* 2022.

[11] N. Petrović, M. Cankar and A. Luzar, "Automated Approach to IaC Code Inspection Using Python-Based DevSecOps Tool," in *2022 30th Telecommunications Forum (TELFOR)*, Belgrade, 2022.

[12] "IaC Scan Runner," [Online]. Available: https://xlab-si.github.io/iac-scanner-docs/02-runner.html. [Accessed 21 November 2022].

# APPENDIX: IaC Scan Runner – current database model

## Database model

The underlying database enabling persistence of relevant data is a Mongo DB document store, which is depicted as entity-relationship diagram as shown in Figure 17.

As it can be seen, the scan results are organized by projects, which can be created by users. For each of the project, the relevant info consists of the following:

1) **userid** [optional] – identifier of a user who created the scan project. Relevant only for multi-user environment [still not implemented]

2) **projectid** – unique project identifier

3) **time** – the moment when the project was created

4) **active_config** – a reference to identifier of a configuration, as each project can have a configuration assigned.

At one point in time, project can have up to one project configuration active. The relevant project configuration info consists of the following:

1) **configid** – identifier of a configuration file

2) **parameters** – a dictionary of tool-specific parameters, such as secrets (username, password or token) for scan checks that require additional configuration

**3) checklist – the list of checks that are enabled within a scan project.**

When it comes to individual scan results, for each of them the following info is relevant:

1) **scanid** – identifier of a scan task

2) **projectid** [optional] – identifier of a project to which particular scan execution belongs. Can be left out or empty in non-persistent mode, then default checks will be considered. Depending on this value, various combination of scan workflow tools will be taken into account for execution.

3) **archive** – name of the user-provided IaC archive

4) **execution-duration** – time spent for scan workflow execution

5) **outcomes** – list containing the results of the performed scan workflows against the IaC archive

5.1 name – the reference the particular scan workflow tool

5.2. status – whether the check reported problems or no

5.2 files - list of files scanned using that particular tool

5.3 log - contents of particular scan tool log in raw textual format

6) **verdict** – the final outcome of the scan, dependent on individual scan results

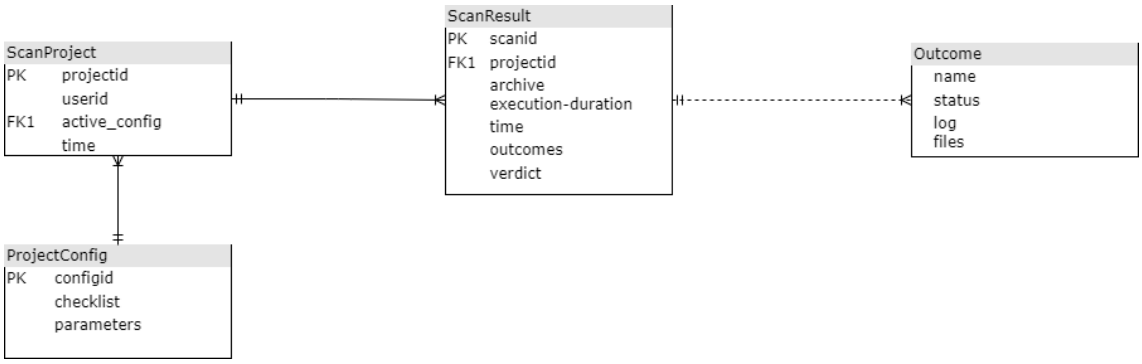


*Figure 17 ER diagram of underlying database model for scan result persistence and project configuration*

## Internal details of the IaC Scan Runner implementation

IaC Scan Runner directory contains the following files and folders, which are summarized together with their role in Table 10 Environment variables for persistence settings enable/disable.

*Table 10 Environment variables for persistence settings enable/disable*

| Name | Type | Description |
|---|---|---|
| **src** | Folder | Folder which consists of all the Python source code files (.py) that make the IaC Scan Runner tool |
| **tools** | Folder | Contains the executable files for distinct check tools that are compatible with the current scan workflow. Its content is generated when all the dependencies are downloaded either from requirements.txt or while building Docker image. |
| **outputs** | Folder | Directory where the files generated during the execution of scan workflow are placed. These folders are generated as part of install_checks.sh sequence of operations. Furthermore, it contains three sub-directories:<br>1) logs – raw logs directly from check tools, one folder with textual files per scan execution<br>2) json_dumps – outcomes summarized in form of JSON file, one file per scan<br><br>generated_html – HTML summaries of the scan results, one file per scan |
| **examples** | Folder | A collection of Python scripts which show how the interaction to various aspects of IaC Scan Runner is performed via its REST API, such as project creation, assigning configuration to a new project and performing scan. |
| **config** | Folder | A set of configuration files (YAML and others) required for various scan workflow tools. |
| **requirements** | Txt | List of IaC Scan Runner dependencies which must be installed before running |

| | | |
|---|---|---|
| **install_checks** | Sh | A shell script that performs a sequence of steps that aim to prepare IaC Scan Runner for scan task execution, such as checking if all the required folders exist and their creation if missing, together with setting the relevant environment variables |
| **Dockerfile** | Text without extension | Describing how Docker image for containerized IaC Scan Runner is created |
| **docker-compose** | YAML | A configuration file defining the services required for IaC Scan Runner, specifying the ports they use, exposing them to the outer world and setting the corresponding environment variables. |