PLEDGER

D3.4 Performance Measurements and classification tools II

Document Identification			
Status	Final	**Due Date**	31/05/2022
Version	1.0	**Submission Date**	01/06/2022

Related WP	WP3	**Document Reference**	D3.4
Related Deliverable(s)	D2.2, D2.3, D3.1	**Dissemination Level (*)**	PU
Lead Participant	ENG	**Lead Author**	Gabriele Giammatteo (ENG)
Contributors	ENG	**Reviewers**	Roi Sucasas (ATOS)
			Orfeas Voutyras (ICCS)

Keywords:
Benchmarking, Evaluation, Performance, Prototype, Architecture, Demo

This document is issued within the frame and for the purpose of the PLEDGER project. This project has received funding from the European Union's Horizon2020 Framework Programme under Grant Agreement No. 871536. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

(*) Dissemination level: **PU**: Public, fully open, e.g. web

Document Information

List of Contributors	
Name	Partner
Gabriele Giammatteo	ENG
Francesco Iadanza	ENG
Keven Thomas Kearney	ENG

Document History			
Version	Date	Change editors	Changes
0.1	01/04/2022	Gabriele Giammatteo (ENG)	Initial TOC and first updated version from D3.1
0.2	27/04/2022	Gabriele Giammatteo (ENG), Francesco Iadanza (ENG), Keven Thomas Kearney (ENG)	Section 2 and section 3 provided
0.3	06/05/2022	Gabriele Giammatteo (ENG)	Section 4, Introduction, and Conclusions improved
0.4	11/05/2022	Gabriele Giammatteo (ENG)	Finalization
0.5	16/05/2022	Roi Sucasas (ATOS)	Internal review
0.6	18/05/2022	Orfeas Voutyras (ICCS)	Internal review
0.7	23/05/2022	Gabriele Giammatteo (ENG)	Internal review comments accepted
0.8	31/05/2022	Carmen San Román (ATOS) Lara López (ATOS)	Quality assurance check and Project Coordinator review
0.9	01/06/2022	Gabriele Giammatteo (ENG)	Project Coordinator comments addressed.
1.0	01/06/2022	Lara López (ATOS)	Final version to be submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Gabriele Giammatteo (ENG)	23/05/2022
Quality manager	Carmen San Román (ATOS)	31/05/2022
Project Coordinator	Lara López (ATOS)	01/06/2022

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
API	Application Programming Interface
AWS	Amazon Web Services
BSD	Berkeley Software Distribution
CIS	Center for Internet Security
CLI	Command Line Interface
CPU	Central Processing Unit
Dx.y	Deliverable number y belonging to WP x
FC	Functional Component
FR	Functional Requirement
GNU	GNU is Not Unix
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IdP	Identity Provider
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
Mx	Project Month x
MVP	Minimum Viable Product
NFR	Non-Functional Requirement
REST	REpresentational State Transfer
SSH	Secure Shell Protocol
SUC	System Use Case
TCP	Transmission Control Protocol
Tx.y	Task number y belonging to WP x
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VPN	Virtual Private Network
WP	Work Package

Executive Summary

This document accompanies the delivery of the final prototype of the Pledger's Benchmarking subsystem developed in Task 3.1 (T3.1) "Performance Measurements and Classification". The main responsibility of the Benchmarking subsystem is to provide performance data for the infrastructures managed by the Pledger system to better characterise them and to optimise the orchestration and deployment decisions for the applications.

This document is an update of the deliverable "D3.1 - Performance Measurements and classification tools I" [1] and includes all the new work performed in task T3.1 in the second project period. The work finalized the implementation and improved the prototype realized during the first period based on the "Benchmarking Suite" asset brought and developed in the project by Engineering Ingegneria Informatica (ENG).

The main achievements in the second period are mainly: a) the development of Docker executor, b) the integration of new benchmarking tools c) the development of an analytics component, d) the development of measuring and storage mechanisms for system-level metrics and e) minor improvements in multiple components (e.g., visualization dashboards, integrated tools and workloads, execution results parsing and metrics computation). The work has been performed under the guidance of the analysis and design performed in Work Package 2 (WP2) and reported in deliverables "D2.2 - Pledger Requirements Analysis" [2] and "D2.3 - Pledger Overall Architecture" [3].

The Benchmarking Suite allows users to benefit from performance evaluations without the effort and the expertise needed to execute the tests and analyse the results manually. The main functionalities implemented are: a) the definition of a library of benchmarking tests representative of different types of applications, b) the automated execution of the workloads on target infrastructures (supporting both Cloud and Edge infrastructures), c) the measurement and the storage of metrics during the executions, d) the computation and publication of evaluation reports, e) the management of the metrics and evaluations to keep them always up-to-date.

The Benchmarking Suite is a cloud native, Kubernetes-based, microservice-oriented application that is constituted by multiple containerized services communicating over network. The Benchmarking Suite is able to scale-in/out its components by communicating with the underlying Kubernetes cluster to adapt to the application load. The application integrates with several other third-party cloud-native technologies like Keycloak, Grafana, Kafka, MongoDB, and InfluxDB, to realize some of its functionalities. The programming languages used for the Benchmarking Suite are Python (for the backend components), Golang (for the InfluxDB proxy), and JavaScript (using the Vue.js framework for the Graphical User Interface (GUI)).

The Benchmarking Suite integrates with the Pledger Configuration subsystem component listening to changes in the infrastructures registered to the system to automatically schedule/update execution of benchmarks on those infrastructures. It is also integrated with the Recommender component by sending performance evaluation reports. Finally, the Benchmarking Suite is also used in the Pledger system in the learning process of the AppProfiler component.

The Benchmarking Suite is released under the open-source license Apache version 2.0 and the source code is available at https://gitlab.com/pledger/public/benchmarking/. The tool provides also an online user documentation available at https://benchmarking-suite.readthedocs.io/.

1 Introduction

1.1 Purpose of the document

This document accompanies the final release of the components in the Benchmarking subsystem developed in the context of project's Task "T3.1 - Performance Measurements and Classification". The document presents the main functional and technical aspects of the prototype as well as guidelines to use it. This document updates the deliverable "D3.1 - Performance Measurements and classification tools I" [1] released at month 18 (M18).

1.2 Relation to other project work

The work reported in this document has been performed in task T3.1 under the guidance of the analysis and design performed in Work Package 2 (WP2). The Benchmarking subsystem is highly coupled with the other components developed in WP3 as well as with the Recommender in WP4 since they will use the benchmarking results to compare infrastructures and provide deployment suggestions and optimizations. Benchmarking components also interact with the Configuration subsystem (WP4) to identify infrastructures to test. Finally, Benchmarking components are integrated, deployed and evaluated in the context of WP5.

1.3 Structure of the document

This document is structured in 4 additional major sections:

- **Section 2** presents the functional description of the tools in alignment with the overall architecture of Pledger.
- **Section 3** presents the technical description of the tools (data model, interfaces, and component architecture).
- **Section 4** presents the installation and usage requirements and guides.
- **Section 5** concludes the document with a summary of the work done and the future work.

In addition, **Appendix I** and **Appendix II** provide more details on the Benchmarking Suite APIs and GUI respectively.

1.4 Progresses in task T3.1 in the second project period

During the second period of the project, a number of activities has been executed in T3.1 to complete the work done in the first period and add the new functionalities expected for the second period.

In general, while in the first period the work done mainly aimed at refactoring and adapting the pre-existing Benchmarking Suite asset to make it work seamlessly with the other Pledger core components and testbeds, in the second period the work was focused mainly on improving and finalizing the existing integrations, developing mechanisms to analyse the metrics collected, improving usability and improving the support for additional infrastructure and benchmarking tools and domains.

In particular, the main activities and achievements have been:

- **implementation of a Docker Executor** and improvement of Kubernetes and Cloud executors for the integration with the project's testbeds (section 3.4);
- **integration of new benchmarking tools** (i.e., CIS Docker, CIS Kubernetes, CloudSuite In Memory Analytics, Perftest RabbitMQ, SciKit Learn, Sysbench MySQL and Tensorflow) to enlarge the assessment for more application types and domains (section 3.2);
- **tune of the existing benchmarking tools** definition and parameters (section 3.2);

- **definition of higher-level metrics** like the *PerformanceIndex* metric and monthly aggregations of the application-level metrics collected during the executions (section 3.3.3);
- **development of an analytics component** to automate the computation of higher-level metrics (section 3.1);
- development of mechanisms **to monitor and collect system-level metrics** during the executions and make them accessible through Visualization component (section 3.3.2);
- **improvements to the Visualization and GUI components** to improve usability of the Benchmarking Suite.

1.5 Glossary adopted in this document

- **Pledger core system**. The system that aggregates all the core results and developed components of the project. Corresponding components result in the "minimum viable product" (MVP) of Pledger, the system that will be freely available to users and other projects and initiatives. Most of the core components represent the exploitable part of the project and contain all the main features that will deem Pledger successful as a project.
- **Pledger core subsystems**. The subsystems that belong to the Pledger core system.
- **(Functional) Component**. A module / component of the system offering a specific functionality.
- **Asset**. A tool already available by one of the partners of the consortium for the materialisation of functional components or subsystems.
- **(Functional) Subsystem**. A group of interrelated functional components.

2 Functional description

This section introduces the main functionalities and components of the Pledger Benchmarking core subsystem developed within Task "T3.1 – Performance Measurements and Classification". The Benchmarking subsystem is one of the core subsystems (Figure 1) identified in deliverable "D2.3 Pledger Overall Architecture" [3], which acts as the roadmap for all development and integration tasks of the project.

This section provides an update to the functional description already provided in the section 2 of the deliverable "D3.1 - Performance Measurements and classification tools I" [1] delivered in May 2021. In particular:

- the overall objective, main functionalities, principles and improvements beyond the State of the Art of the Benchmarking subsystem are presented in section 2.1;
- the description of the functional modules identified within the Benchmarking subsystem are presented in section 2.2;
- the description of the interactions with Pledger modules external to the Benchmarking subsystem are presented in section 2.3;
- the progress of the development activities in the task T3.1 in the first and second period of the project in relationship with the project's requirements and the recommendations from the first project review are reported in section 2.4.

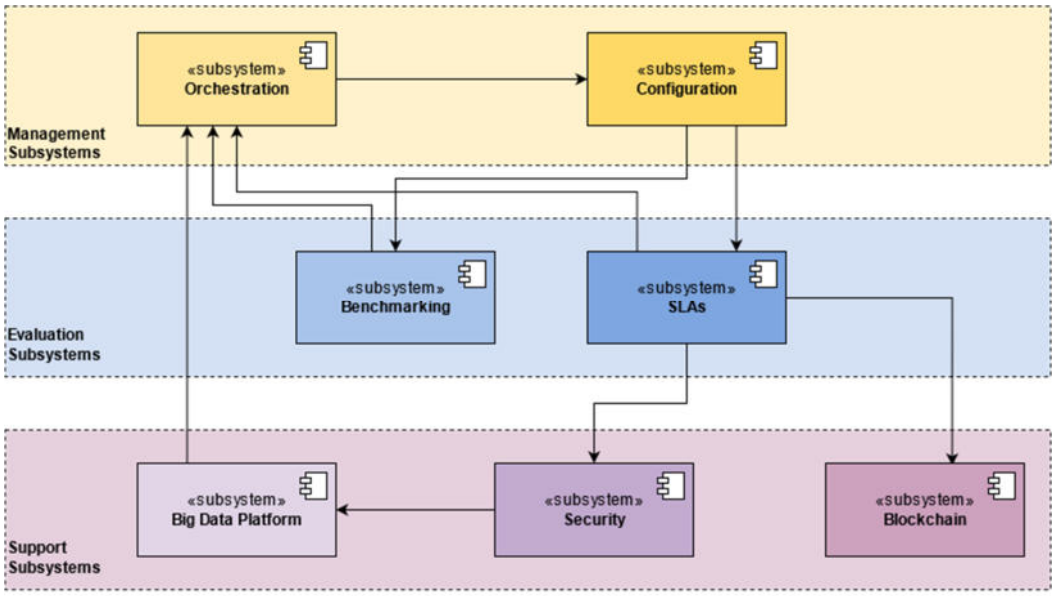

Figure 1: Pledger's Core Subsystems

2.1 Objective of the Benchmarking subsystem

The overall objective of the Benchmarking subsystem is to provide up-to-date performance evaluations for all the infrastructures managed by Pledger to better characterise them and to optimise the deployment and orchestration of the applications.

This is achieved through the following list of functionalities identified in deliverable "D3.1 - Performance Measurements and classification tools I" [1] and reported below:

- define and maintain a library of benchmark tests (workloads) representative of different types of applications (e.g., CPU intensive, network intensive, database, machine learning) that can simulate the system resources usage patterns of real applications;
- execute the workloads on the managed infrastructures supporting both cloud and edge resources and interfacing with different technologies (e.g., Kubernetes, Openstack, Docker, bare metal);
- measure application-level and system-level metrics during the execution of workloads and compute performance indexes as well as higher level indicators of the performance of the infrastructures;
- keep performance data always fresh and updated by repeating the executions a) periodically to track performance changes, b) on detected changes in the infrastructure configuration and/or c) on demand scheduled by the user;
- publish performance reports to the other Pledger modules to be used for decision-making.

The main principles followed in the design of the benchmarking functionalities are a) the automation of all the steps of the benchmarking process (from the discovery of the infrastructure resources to the configuration, scheduling, execution of tests and finally the collection and analysis of the results) and b) the analysis of the execution results to produce synthetic and more human friendly performance indicators. **This allows Pledger users to benefit from performance evaluations without the effort and the expertise needed to execute the tests and analyse the results manually.** The Benchmarking subsystem is highly integrated also with the rest of Pledger components (see section 2.3) to further increase the level of automation and self-configuration of the benchmarking process within the Pledger system (Figure 2, Table 1).

The work aims at providing an innovative benchmarking service that goes **beyond the State of the Art** in some aspects, by:

- providing a **complete automation** of the benchmarking process supporting both Cloud and Edge technologies and multiple different benchmarking test suites. To achieve that, abstract models for workloads and infrastructures has been defined as well as executors and connectors able to interact with different infrastructure technologies;
- **analysing performance metrics** collected during the execution of tests with the goal of a) making them uniform and comparable across tests and infrastructures and b) computing **higher level metrics**, like overall performance assessment and stability of an infrastructure. User-friendly tools that easily visualize and compare performance of different infrastructures to support the decision process are also provided;
- providing a **comprehensive test suite for Edge** computing by adapting Cloud workloads, integrating Edge benchmarking tools from the community and, supported by project use cases, selecting or creating new tests representative of typical Edge use cases;
- increasing **security of the benchmarking process**, by checking the code being executed and setting quotas and restrictions for the executions.

2.2 Description of the functional components

The objective and the functionalities identified in section 2.1 guided the design of the Benchmarking subsystem and the identification of its functional components.

The functional components have been already identified in deliverable “D3.1 - Performance Measurements and classification tools I” [1] (in section 2.2) released in the first period in May 2021 and they are reported in Figure 2 and Table 1 for convenience.

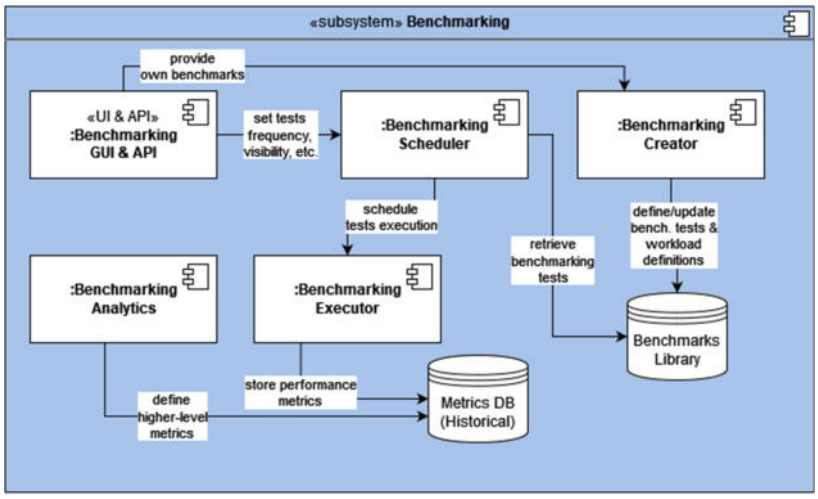

Figure 2: Benchmarking subsystem components

Table 1: Benchmarking subsystem Functional Components (FCs)

ID	Component	Functionality
FC.3.1	**Benchmarking Creator**	The Benchmarking Creator is responsible for the creation of user-defined benchmarking tests to be included in the Benchmarking Library.
FC.3.2	**Benchmarking Scheduler**	The Benchmarking Scheduler is responsible for the scheduling of the executions of benchmarking tests on the infrastructures. Each schedule managed by this component executes a set of tests on a given infrastructure with a given frequency.
FC.3.3	**Benchmarking Executor**	The Benchmarking Executor is responsible for execute the tests. It connects to the target infrastructure, creates the needed resources with the required configuration, install and execute tests and collects the results.
FC.3.4	**Benchmarks Library**	The Benchmarking Library is a database of benchmark tests that include, for each test, the tools, the execution instructions and the result parsers. The library contains both generic tests available to all use cases, as well as tests defined by the user and used only for her specific use case.
FC.3.5	**Benchmarking Metrics DB**	The Benchmarking Metrics DB stores the benchmarking results of the executed tests. It provides an interface to query the results and perform mathematical operations over the results as well as returning the results in a time-series format.
FC.3.6	**Benchmarking Analytics**	This component analyses metrics in the Metrics DB and computes new metrics based on the ones already stored. The objective is to create new derived metrics that are stored back in the Metrics DB. Examples of metrics are: weekly average performance for workloads on the different providers.
FC.3.7	**Benchmarking GUI & API**	The GUI consists of a web application that provides access to all functionalities of the Benchmarking Suite. The API component realizes a REST interface on top of the Benchsuite DB. It is currently mainly used by the GUI to fetch/store the Benchmarking Suite configurations.

In the Pledger's scenarios, the benchmarking process is triggered by changes in the infrastructure configuration notified by the Configuration subsystem (e.g., a new infrastructure is added or modified). The **Benchmarking Scheduler** component automatically schedules the execution of a set of benchmarking tests from a library of default tests in the **Benchmarks Library**. Beside the automatic settings, the user can adjust and customize the behaviour of benchmarking from an ad-hoc GUI (the **Benchmarking GUI** component) that can be used to activate/deactivate default tests, define new tests, set frequency and visibility of results, etc.

The **Benchmarking Creator** component allows to create new (or update existing) workload definitions in the Benchmarking Library. This functionality is intended not only for administrators of the system, but also for expert users that want integrate a new third-party benchmarking tool.

In order to execute the tests, the **Benchmarking Executor** connects to the external infrastructures using the credentials available in the Configuration subsystem and executes the benchmarking workloads measuring the response of the system. The performance metrics are collected, aggregated, and made available for graphical analysis (as well as through REST API) in the **Metrics DB**. The **Benchmarking Analytics** module analyses the metrics from the Metrics DB to produce higher level, human friendly performance indicators and statistics. In addition, a subset of metrics is shared with the Configuration subsystem to make them available to the other Pledger components (e.g., Recommender).

2.3 Interactions with other Pledger modules

As introduced in section 2.1, the integration of the Benchmarking subsystem with other Pledger modules aims at automatizing the triggering of the performance evaluation process and the sharing of its results in the system.

The interactions with the other Pledger components have been identified and described already in the first project ("D3.1 - Performance Measurements and classification tools I" [1]). **During the second period, the data models for the exchanged messages and the integration mechanisms have been improved and refined.** This section lists the four integrations between Pledger modules and the Benchmarking subsystem (see Figure 3).

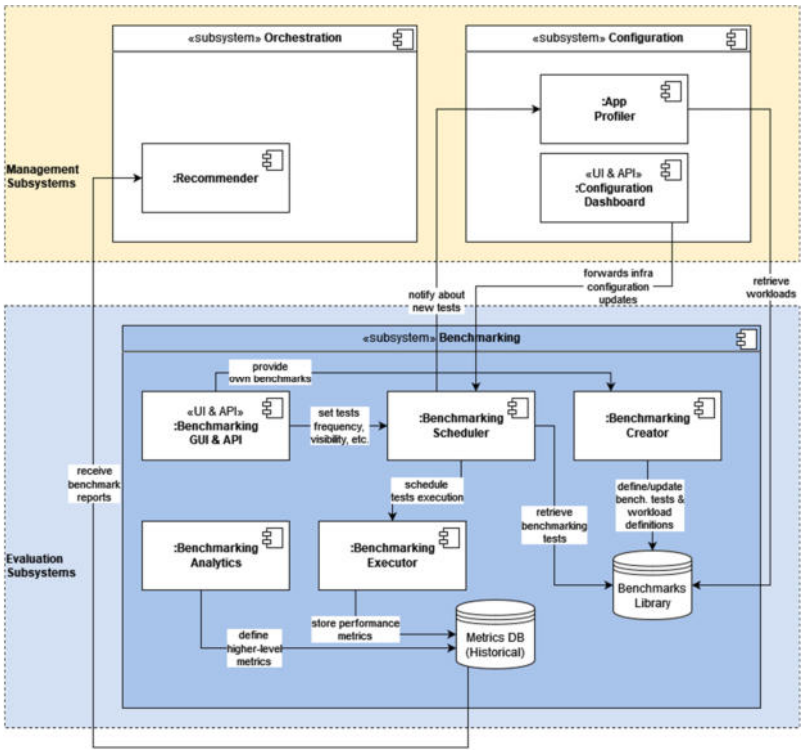

Figure 3: Benchmarking subsystem interaction with other subsystems

Configuration Subsystem. The Configuration subsystem keeps the main configuration of infrastructures and applications managed by the Pledger system. The Benchmarking Scheduler reads the configuration of infrastructures registered in the system with the objective of evaluating their performance by scheduling the execution of benchmark tests. Data retrieved from the Configuration subsystem includes: infrastructure access information (e.g., endpoints, credentials), the topology of the infrastructures (i.e., nodes and their characteristics), benchmarking preferences (i.e., enabled or not, quotas and frequency of execution, scope of the results). The Benchmarking Scheduler also listens for notifications from the Configuration subsystem about changes in the existing infrastructures, to update its execution schedules.

Recommender. The Recommender is the component that, analysing the metrics received by the other modules in the system, provides suggestions on actions to maintain or improve the performance and QoS of the applications managed by Pledger. It receives the results of the performance evaluations (i.e., benchmarking reports) generated by the Benchmarking Metrics DB component. The reports indicate the performance measured for each executed workload on each enabled infrastructure along with details on the workloads and executions parameters and all the application-level and system-level metrics collected. The reports allow the Recommender to estimate on which infrastructure an application that resembles a given workload is capable of achieving better performance.

AppProfiler. The AppProfiler needs to profile the executions of benchmarking workloads to train its internal model. In this way, when a user application is profiled, it will be able to match the application with one particular benchmark workload that has similar characteristics of the user application. To train the AppProfiler with the benchmarking workloads, all the workloads available in the Benchmarking Library are executed in a particular session where the Benchmarking Executor is configured to notify the AppProfiler when the execution of a new test is about to start. The notification contains the name of the workload. In this way, the AppProfiler can start to profile the execution and, when it is completed, store the profile associated with the workload name.

Big Data & Communication Platform subsystem (StreamHandler). The StreamHandler is the software bus system used in the Pledger system. It is used to technically realize all the integration mentioned above between the Benchmarking components and the other Pledger modules. The usage of the StreamHandler allows to decouple all the components of the system and make the components interoperate just through the exchange of messages in a standard format. As it is used as an integration pattern through a public subscribe mechanism by various components of Pledger, the corresponding connections (arrows) are not provided.

2.4 Requirements implementation

The implementation of the benchmarking functionalities started from a pre-existing asset owned by Engineering Ingegneria Informatica S.p.A. (ENG) called “Benchmarking Suite” (also known as Benchsuite) evolved and adapted in order to fit the needs of the benchmarking process in the Pledger project. The development work proceeded by following the requirements identified early in the project in deliverable D2.2 [2] plus the recommendations received during the interim project review meeting.

Table 2 summarizes the main development work done to satisfy the project requirements and the first project review recommendations along with indications on whether it has been done in the first or second period of the project.

Table 2: Work done in relationship with the Pledger requirements

Software Module/Activity	Description	Pledger Req.
Kubernetes executor	This executor allows to automate the execution of benchmarking tests on Kubernetes clusters. It has been developed to support the	FR.64

Software Module/Activity	Description	Pledger Req.
	benchmarking of some of the project's testbeds (e.g., ENG and i2CAT testbeds). The first implementation has been delivered during the first period and improved and finalized during the second period.	
Docker executor	This executor allows to automate the execution of benchmarking tests on Docker clusters or machines that runs the Docker daemon. It has been developed to support the benchmarking of Edge nodes in the project that run Docker (e.g., FILL Edge nodes) and for the training of the AppProfiler models. This module has been developed in the second project period.	FR.64
Analytics module	This is a new module to process raw metrics from benchmarking results and compute higher level metrics. At the moment, computation of the *PerformanceIndex* metric, the average performance, the standard deviation and the approximate entropy over the last month. This module was implemented during the second project period.	FR.68
Workloads definition	A new definition of workload was needed to support Docker and Kubernetes environments. The previous one (based on scripts to be executed via SSH) was adequate only for VMs and bare metal machines, but not for Docker and Kubernetes. The new definition is based on containers (each one with a different role in the test) that can be arranged in flexible a topology. The activity started during the first period and has been fully completed during the second period.	NFR.08
Workloads migration	The new definition of workload required a migration of the workloads from the old definition to the new one. This was not just a translation to the new format, but also needed the containerization of the benchmark tools. This process started during the first period and finalized during the second period.	FR.67
GUI Improvements	Changes and improvements of the GUI component in order to adapt to the new workload definition, to improve the visualization of execution results and logs. This process started during the first period and finalized during the second period.	FR.63 FR.69
Kubernetes deployment	A Helm chart has been developed to deploy the Benchmarking Suite in a Kubernetes cluster (see section 4). This improved a) the deployment in the project's testbeds, b) the possibility to automate the deployment in the CI/CD pipeline and c) the improvement of the robustness, scalability and resilience of the tool. This required, beside the development of the chart, also the containerization of some modules and the modification of how the different modules load the initial configuration, expose their APIs and interacts with the other modules. This activity has been fully implemented during the first project period.	NFR.06

Software Module/Activity	Description	Pledger Req.
Security	Oauth2.0 authentication has been added to access the Visualization module and the Metrics DB. Before Pledger, it was available only for the GUI. This required the development of the Metrics DB proxy. Other improvements have been done like encryption of infrastructure credentials in the Benchsuite DB. This activity has been fully implemented during the first project period.	NFR.09
Multitenancy	A proper authorization model has been developed in the tool to ensure users can only access the data they are authorized to access. This required several changes mainly in the API and Metrics DB Proxy components. Also, the GUI and the executor components have been adapted. This activity has been fully implemented during the first project period.	FR.65 FR.66
Metrics DB	The previous version of the Metrics DB was based on MySQL. It has been replaced by InfluxDB that provide more performance in storing and querying data. Moreover, it provides a richer interface for querying data including mathematical and aggregation functions. This activity has been performed during the first project period.	FR.68 NFR.07
Visualization	The Visualization module is a new module based on Grafana with a custom dashboard to visualize metrics from the Metrics DB. It allows to aggregate and filter data by time range, workload and infrastructure. It has been fully integrated in the Benchmarking Suite from the deployment, authentication and user management point of views. The work on this activity started during the first period and continued during the second period with the improvement of the existing dashboards and the addition of new ones.	FR.63 FR.69
Configuration subsystem Integration	Integration with the Pledger's Configuration subsystem has been developed (through the StreamHandlder bus component). It allows to automatically configure the Benchmarking Suite to test the infrastructure registered in the Pledger system. This integration was developed during the first project period and improved during the second period.	SUC.18
Recommender Integration	Integration with the Pledger Recommender component has been implemented (through the StreamHandler). It allows the Recommender to receive benchmarking reports from the Benchmarking Suite containing performance evaluations for different workloads on the infrastructures registered in the Pledger system. This integration was developed during the first project period and improved during the second period.	FR.69 SUC.21
System-level metrics	This new functionality allows to monitoring system-level metrics (e.g. CPU and memory utilization) during the execution of benchmarking tests and use them to compute better performance assessments and predictions. The implementation involved work on multiple Benchmarking Suite modules (i.e., Executor, Metrics DB, Analytics and Visualization). This functionality has been developed during the second project period.	FR.68 Review Recom.

3 Technical description

As analysed in deliverable D2.2 [2], the benchmarking subsystem is realized by combining specialized benchmarking tools with the Benchmarking Suite (also known as Benchsuite) asset brought and developed in the project by Engineering Ingegneria Informatica (ENG), initially developed in the previous EU research project CloudPerfect [4].

The Benchmarking Suite focuses on the orchestration and automation of the benchmarking process. In Pledger, it has been evolved and adapted to the needs of the project's infrastructures and use cases. In particular, the main improvements are related to the new support for Kubernetes and Edge nodes the extended metric analysis and the extended support for 3rd party benchmarking tools.

This section provides an overview of the architecture, design, and implementation of the Benchmarking Suite in section 3.1 and a set of sections focused on some specific topics: the integrated 3rd party benchmarking tools (section 3.2), the metrics collected and computed from test executions (section 3.3) and the infrastructures supported (section 3.4).

A description of the technologies used by the Benchmarking Suite, its software dependencies, and how is deployed and integrated with Kubernetes can be found in the sections 3.2 and 3.6 of the deliverable "D3.1 - Performance Measurements and classification tools I" [1].

3.1 Design and internal architecture

This section reports the internal architecture and the design choices of the Benchmarking Suite.

Excepting for the Analytics component, all the other architectural concepts and components have been already identified in the first project period and have been introduced in the first version of "D3.1 - Performance Measurements and classification tools I" [1] (delivered in May 2021). They are reported in this section for the sake of completeness and to provide an overview of the final tool architecture.

The Benchmarking Suite manages the following main concepts (depicted in Figure 4):

- **Provider**: represents a target infrastructure to benchmark. Multiple technologies are supported, like Openstack and VMWare Cloud platforms, Docker and Kubernetes clusters and bare metal machines or Virtual Machines with SSH access.
- **Workload**: represent a benchmark tool that will be executed on a Provider to test its performance. The workload can be defined in various format like bash scripts, Docker containers or with a topology of multiple containers.
- **Schedule**: represents a benchmarking session. It includes a Provider and one or more Workloads to be executed on the provider. It also includes a time interval that represents the frequency at which the session will be re-executed.
- **Execution**: represents a single execution of a Workload on a Provider.
- **Metric**: represents a metric measured during one Execution. It is characterised by a name, a value, a timestamp, a set of metadata key-value tags (e.g., *providerId*, *workloadId*, *visibility*) and a visibility scope (i.e., private, organization and public).

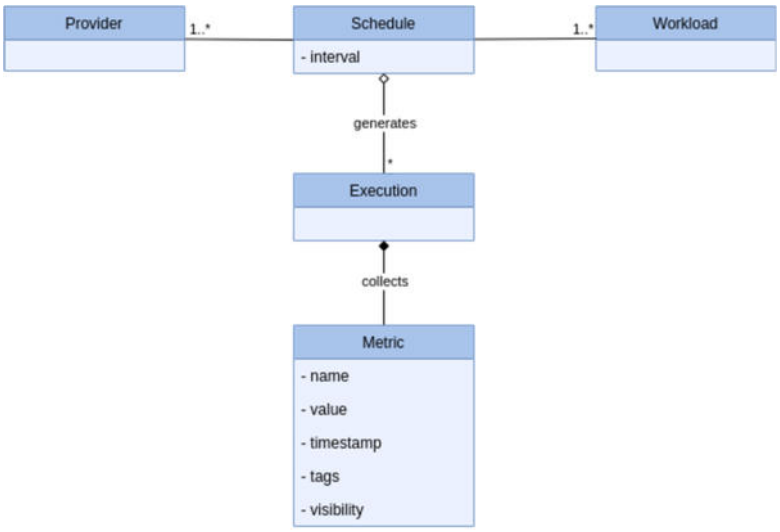

Figure 4: Benchmarking Suite domain model

Figure 5 shows the internal architecture of the Benchmarking Suite tool. For each component, a brief description of its objective, design, interactions and some implementation choices are provided in the remaining of this section.

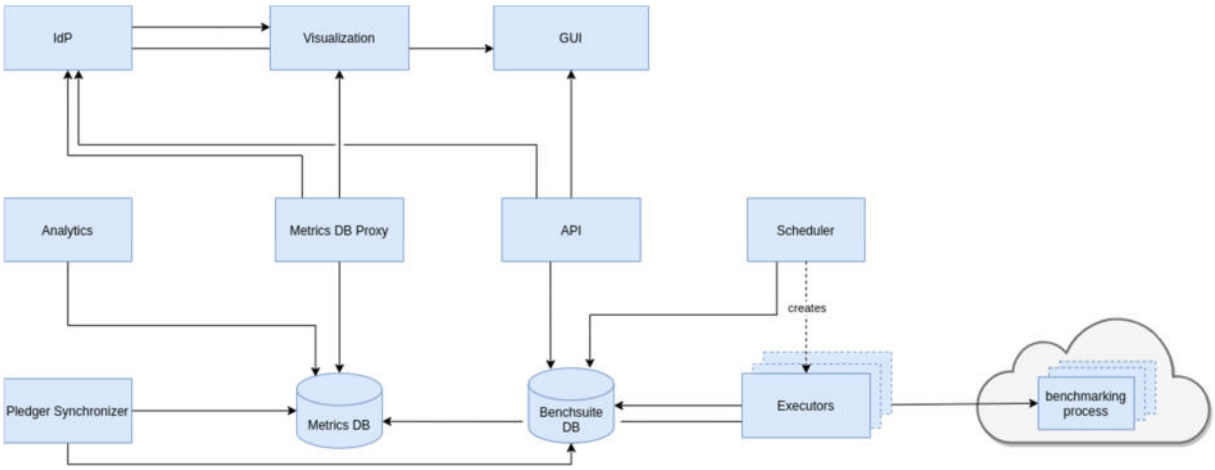

Figure 5: Benchmarking Suite Architecture

Benchsuite DB. This component maintains the main configuration of the Benchmarking Suite tool. It stores all the domain model objects of the Benchmarking Suite (i.e., providers, workloads, schedules and executions) excepting for the metrics which are stored in the Metrics DB. Attributes that are considered secrets (e.g., credentials of providers) are stored encrypted in order to avoid their disclosure in case of unauthorized accesses. The component is realized using a MongoDB [5] database. It has been preferred over relational databases for the flexibility allowed in the objects schema. This is very useful characteristic in the case of the Benchmarking Suite because a) providers and workloads objects can have different attributes depending on their type and b) the schema of the objects can be easily updated during the development.

Metrics DB. Stores the metrics collected from the executions of the tests. Given the nature of the metrics data, a time series database is used to efficiently store and query the metrics. Time series databases are particularly optimized to make queries that makes various aggregation of numeric data over a time range. This is the most common type of queries used by the Benchmarking Suite (e.g., find the provider that

maximize a given metric in the last month). The technology selected to implement this component is InfluxDB [6]. It has been selected over other time series databases (e.g., Prometheus [7]) for its greater flexibility in the data model (e.g., possibility to store more fields for the same metric, possibility to use a custom timestamp when storing the metrics).

Metrics DB Proxy. This component proxies all the queries directed to the Metrics DB and enforces the visibility of the results making sure that only metrics visible to the user that made the request are returned. It identifies the user and organization by looking at the authentication tokens in the request and enriches the query adding clauses that limit the results to the ones that have a visibility that matches the user' scopes. Then it forwards the request to the Metrics DB. This component also solves the lack of support for OAuth2.0 in InfluxDB: authentication and authorization are done by the Metrics DB Proxy before forwarding the request to the Metrics DB. This component has been developed using Golang to provide a high performance, given the fact that it is executed at every request to the database.

Scheduler. This component schedules the executions of tests looking at the schedules stored in the Benchsuite DB. The time interval specified in each schedule is used to set a time for the execution**.** When it is the time to execute a schedule, the Scheduler launches an Executor process configured with the schedule's data. The Scheduler supports launching of Executor processes as a) local processes, b) Docker containers and c) Kubernetes jobs. The choice between the three mods depends on the deployment schema of the Benchmarking Suite instance (e.g., if it is deployed in Kubernetes, it will start Executors as Kubernetes jobs).

Executor. The Executor is the component responsible for the execution of tests. It is invoked specifying one provider and a list of workloads to execute. The Executor connects to the provider, creates the required environment (e.g., a VM, a Docker container) and then proceed with the execution of the test specified by the workload. The Executor is also responsible to monitor the system during the execution of tests. Depending on the source of the monitoring data, this could also need to deploy/start monitoring probes. At the end of the execution, the output and the metrics of the execution are collected from the provider and the environment is cleaned-up. Finally, metrics collected are stored into the *Metrics DB* and the execution status is stored in the *Benchsuite DB*.

API. The API component realizes a REST interface on top of the Benchsuite DB. It is currently mainly used by the GUI to fetch/store the Benchmarking Suite configuration. It is integrated with the Identity Provider (IdP) component to authenticate each request. Depending on the user identity and scopes, data returned will be filtered to show only objects visible to the requesting user and operations will be authorized only if the user has permissions to perform them. The complete list of operations of the API is available in section .

GUI. The GUI consists of a web application (realized with the Vue.js [8] framework) that provides access to all functionalities of the Benchmarking Suite: a) creation/modification/removal of providers, workloads and schedules and b) visualization of the status, logs and metrics for the executions. The only operations that are not included in the GUI are the user management operations. At the moment, these operations are performed directly in the IdP component. The GUI is integrated with the IdP for the login/logout of the user. Some screenshots of the Visualization tools are presented from Figure 6 to Figure 14 in .

Visualization. This component has the objective of providing a user interface to visualize the metrics and provide basic analysis tools like filtering, charts and tables. This component accesses the *Metrics DB* and visualizes the metrics in a set of customizable dashboards. The component is integrated with the *IdP* to perform authentication of the user and ensure that only data visible to the user (generated by her own executions, by other users in the same organization or public) is visualized. At the moment, this component is based on Grafana [10] with an ad-hoc configuration and customized data sources and dashboards. Some screenshots of the Visualization tools are presented in Figure 11, Figure 12 and Figure 13 in Appendix II – Benchmarking Suite Demonstration.

IdP: The Identity Provider (IdP) component is responsible for managing the users and the roles of the users in the Benchmarking Suite and providers authentication and authorization mechanisms. It is realized using the Keycloak [9] server and the Oauth2.0 protocol for login/logout.

Analytics. This component analyses metrics in the *Metrics DB* and compute new metrics based on the ones stored in the Metrics DB. The objective is to create new derived metrics that are stored back in the Metrics DB. Examples of metrics are weekly average performance for workloads on the different providers. An in-depth description of how this component works and which metrics are computed, is provided in section 3.3.3.

PledgerSynchronizer This component centralizes the integrations with the rest of the Pledger system (see section 2.3). It implements the integration with the StreamHandler to receive messages from the Configuration subsystem about updates to the managed infrastructures (and synchronise the providers and schedules in the Benchmarking Suite accordingly) and send benchmarking reports to the Recommender based on the metrics contained in the *Metrics DB***.**

3.2 Benchmarking Tools

The Benchmarking Suite integrates 3rd party benchmarking tools and relies on them for the execution of the tests and for the extraction of metrics. A Workload definition consists of instructions that the Benchmarking Suite uses to a) install the tool, b) configure it, c) run it and d) extract metrics from its execution (typically from the output of the execution or an output file generated by the tool). Typically, each benchmarking tool can generate different workloads based on the configuration provided, so the same tool can be used in multiple Workload definitions.

The tools currently used in the Benchmarking Suite have been selected for their relevance, popularity, flexibility (in terms of workloads that can generate) and metrics that are measured. During the **second project period new tools have been added** (i.e., CIS Docker, CIS Kubernetes, CloudSuite In Memory Analytics, Perftest RabbitMQ, SciKit Learn, Sysbench MySQL and Tensorflow) and the other ones have been improved. Table 3 summarizes the tools integrated as of April 2022.

Table 3: Integrated benchmarking tools

Tool	Description	License
CIS Docker [11]	A standardized set of **security benchmarks** for Docker defined by the Center for Internet Security. The tool uses an open-source implementation of the benchmarks available at https://github.com/docker/docker-bench-security	Apache License v2
CIS Kubernetes [12]	A standardized set of **security benchmarks** for Kubernetes defined by the Center for Internet Security. The tool uses an open-source implementation of the benchmarks available at https://github.com/aquasecurity/kube-bench	Apache License v2
CloudSuite In Memory Analytics [13]	A benchmark that uses Apache Spark and runs a collaborative filtering algorithm in-memory on a dataset of user-movie ratings	CloudSuite 3.0 License[1]

[1] https://github.com/parsa-epfl/cloudsuite/blob/main/LICENSE.md#software-developed-internally-by-the-cloudsuite-group

Tool	Description	License
DaCapo [14]	A tool for **Java benchmarking**. It consists of a set of open-source, real-world applications with non-trivial memory loads	Apache License v2
FileBench [15]	A very powerful tool able to generate a variety of **filesystem- and storage-based** workloads	CDDL
Iperf [16]	A benchmarking tool to measure the maximum achievable bandwidth on **IP networks**. It provides statistics both for TCP and UDP protocols	BSD 3-clause
Perftest RabbitMQ [17]	A performance testing tool released as part of RabbitMQ. It is used to simulate a **messaging** application that sends and receive messages	Mozilla Public License 2.0 GNU General Public License v2 Apache License v2
Phoronix [18]	A large **collection of workloads spanning different domains** from AI, to audio, graphic, scientific simulation, web servers. All workloads are defined in a standard format and runnable through the Phoronix command line program. The tool has been developed and used for running tests uploaded at the https://openbenchmarking.org/	GNU General Public License v3.0
SciKit Learn [19]	A set of benchmarks for the machine learning framework SciKit Learn. Used to simulate **clustering** and **classification** algorithms	BSD 3-clause
Sysbench CPU [20]	A modular, cross-platform and multi-threaded benchmark tool for evaluating **CPU, memory, file I/O, mutex** performance	GNU General Public License v2
Sysbench MySQL [20]	A modular, cross-platform and multi-threaded benchmark tool for evaluating **MySQL** performance	GNU General Public License v2
YCSB [21]	A **database** benchmarking tool. It has the support for several database technologies and provides a configuration mechanism to simulate different usages	Apache License v2
Tensorflow [22]	A set of benchmarking tests for **convolutional neural networks** implemented in Tensorflow	Apache License v2
TFB [23]	An open-source tool used to compare many **web application frameworks** executing fundamental tasks such as JSON serialization, database access, and server-side template composition. The tool has been developed and it is used to run the tests that generate the results available at: https://www.techempower.com/benchmarks/	TechEmpower open-source license[2]

[2] https://github.com/TechEmpower/FrameworkBenchmarks/blob/master/LICENSE

Each of the tools integrated can generate multiple *Workloads* by varying the input data and/or parameters. As of April 2022, the **14 available tools** are used to generate **65 different workloads**.

A full list of workloads defined for each of these tools is available in the online documentation [24].

3.3 Metrics

Metrics collected during the execution of the benchmarking tests are an essential part of the performance assessment process since they are the input to compute the performance of an infrastructure.

The Benchmarking Suite deals with three different types of metrics:

- **application-level metrics**: generated by the benchmarking tools while they are running;
- **system-level metrics**: target infrastructure resource usage metrics (e.g., CPU and memory usage) collected during the execution of benchmarking tests using external monitoring systems;
- **computed metrics**: generated by the Benchmarking Suite itself as results of the analysis of the other two types of metrics;

All of them concur, at different levels, to the final infrastructure performance assessment. They are briefly presented in the remaining of this section.

3.3.1 Application-level metrics

The external tools that the Benchmarking Suite relies on for the test execution are specialized benchmarking tools (or real applications simulators) that are able to generate metrics during their execution to assess the performance of the system. This type of metrics is specific for each tool because it depends on the application type that the benchmarking tool is simulating and the tool implementation. For instance, YCSB executions generate metrics specific for databases (e.g., *read_ops*, *read_latency*, *insert_ops*), while Iperf generates metrics specific for network (e.g., *transferred_bytes*, *bandwidth*, *packet_loss*).

The Benchmarking Suite is able to parse the metrics generated by an execution from a result file and/or the execution output and store them in the *Metrics DB*. Since the format of the metrics changes from one tool to another, the Benchmarking Suite implements a **customizable parser for the metrics** that can be configured in the *Workload* model. The example below represents two different configurations for the metrics parser: the first one extracts the metrics from the output file based on a text regular expression, while the second one converts the output in a JSON file and parse the metrics using JSONPath expressions.

```
metric-definitions:
 - regex: "sending rate avg: [[send_rate:msg/s:int]] msg\/s"
 - regex: "receiving rate avg: [[recv_rate:msg/s:int]] msg\/s"

[...]

metric-definitions:
 - line-selector: '{"Controls"(.|\s)*'
   json:
    - path: '$.Totals.total_pass'
      name: 'passed'
      type: 'int'
    - path: '$.Totals.total_fail'
      name: 'failed'
      type: 'int'
```

The Benchmarking Suite also supports cases of tools that do not generate any metric during the execution. This can happen for tools that were not born to be benchmarking tools or user-provided tools. In those cases, the Benchmarking Suite generates the ***duration* metric** that corresponds to the overall duration of the execution. The *duration* metric will be meaningful for the performance assessment only if the test runs in such a way that the quicker it finishes the better the performance (e.g., a test that execute a fixed number of operations).

All the metrics collected during this execution phase are stored in the *Metrics DB* and are made available for analysis (by the *Analytics* component) and visualization (through the *Visualization* tool).

The exhaustive list of metrics collected for each tool can be found in the "Benchmark" section of the online documentation [24].

3.3.2 System-level metrics

While the application-level metrics remain the main source to assess the performance of a benchmark execution, monitoring the target system while the test is running can provide additional information about the system behaviour when it is running a given workload. For instance, a given workload might produce the same application-level measure (e.g. number of "insert" operation executed every second) on two different infrastructures, but the two infrastructures could use very different CPU and memory to achieve the same performance. This difference could reveal that one infrastructure uses its resources more efficiently than the other one and, therefore, should be preferred.

In order to measure system-level metrics, the **Benchmarking Suite relies on external monitoring systems** available in the target infrastructure. At the moment, the external monitoring solutions supported are Prometheus [7] and the Docker Stats [25]. The configuration to use the monitoring system (e.g., the URL to access Prometheus APIs) must be provided in the *Provider* configuration (see section 3.1). The *Executor* component, before starting the test execution, is responsible to prepare the monitoring system for the metrics collection. This preparation step is not always needed: for instance, in the case of Prometheus, no preparation is needed, while for collecting Docker Stats metrics in a Kubernetes environment, a probe pod needs to be created on the node where the test will be executed. The *Executor* is responsible for starting and stopping the collection of metrics synchronously with the start and the end of the test execution. The collected metrics are then stored in the *Metrics DB* at the end of the test execution and associated to the other application-level metrics generated by the execution. A specific dashboard for the *Visualization* tool has been developed to chart the system-level metrics (Figure 13).

A challenge, not fully addressed by the current implementation, is the management of the **resolution of the monitoring metrics** (the measurement frequency). This is a parameter that depends on the monitoring system configuration. Some monitoring system are configured with a low resolution (e.g., Prometheus in Kubernetes has, by default, 5 minute resolution to save resources for the measuring and storing), while other systems have higher resolutions (e.g., 1 second) or always take a measure every time it is requested. The resolution should be higher than the duration of the test to have meaningful results. A monitoring system that takes a measurement of CPU every 5 minutes is useless if the test run only for 2 minutes (there will be, at most, only one value generated). In fact, the monitoring system should have a resolution that allows to collect at least 10 values during the test execution (a reasonable threshold to have an idea of the system behaviour). Since the configuration of the external monitoring system is out of the scope of the Benchmarking Suite, the only solution at the moment is to manually change the resolution of the monitoring system or use an alternative monitoring system (for instance Docker Stats in Kubernetes instead of Prometheus).

3.3.3 Computed metrics

Computed metrics are metrics that are not collected during the execution of the tests (like application- and system-level metrics), but they are computed afterwards by the *Analytics* module by processing the application- and system-level metrics stored in the *Metrics DB*. The computed metrics aim at providing an easier and more accurate measurement of the performance of infrastructures.

The **PerformanceIndex** is a metric computed for each execution on the basis of the application- and system- level metrics collected during the execution. Often, just looking at application-level metrics for a given execution does not provide an immediate indicator of the performance of that execution for multiple reasons: some knowledge of the meaning of these metrics is needed, some knowledge on how the tool was configured and how the metric was collected is needed, not all metrics are representative of the overall performance, the overall performance could be the result of the combination of multiple metrics. To overcome, and make transparent, to the final user all these aspects, the *PerformanceIndex* metric is defined based on a formula (configured in the *Workload* model). By convention, the *PerformanceIndex* must be defined to be a monotonic, non-decreasing function (the higher is the value, the better is the performance). In this way, it is possible to immediately compare the performance achieved across two different executions and across different infrastructures. Table 4 shows how the *PerformanceIndex* is defined for the benchmarking tools currently integrated.

Table 4: PerformanceIndex definitions

Tool	PerformanceIndex	Description
CIS Docker	*score*	the achieved score
CIS Kubernetes	*passed / (failed + passed + warns + info)*	the percentage of passed tests
CloudSuite In Memory Analytics	*1 / duration*	inverse of the duration (how fast it ran)
DaCapo	*1 / timed_duration*	inverse of the timed duration (how fast it ran)
FileBench	*ops_throughput*	average number of operations executed per second
Iperf	*received_bandwidth_sum*	bandwidth observed for the connection
Perftest RabbitMQ	*recv_rate*	how many messages are received per second
Phoronix	*test_result*	performance metric computed by Phoronix (the meaning can change depending on the workload executed)
SciKit Learn	*1 / duration*	inverse of the duration (how fast it ran)
Sysbench CPU	*events_rate*	number of times prime numbers between 0 and 20000 are verified each second with X threads
Sysbench MySQL	*latency_avg*	average speed of the operations executed
YCSB	*latency_avg*	average speed of the operations executed
Tensorflow	*img_sec*	how many images processed per-second by the learning process
TFB	*latency_avg*	average speed of the operations executed

The *Analytics* component also computes some time-based aggregations for each application-level metric stored in the *Metrics DB*. In fact, one of the main benefits of the Benchmarking Suite (enabled by the full automation of the test executions) is to allow the re-execution of the same test on the same infrastructure on regular basis. For this reason, there are historical data for each metric that can be processed to extract additional information on the performance (e.g., how much it is stable). The aggregations automatically computed for each metric are:

- **Geometric Mean over last month**: it is the average value of the metric on all the values of that metric measured in all the executions of the last month. It can be used as a more stable, reliable and realistic indicator of the value of that metric instead of the value measured in a single execution;
- **Standard Deviation over last month**: it represents a measure of the amount of variation (from the mean) of measured values of the metric over the last month. The lower it is the standard deviation, the closer the values are to the mean. In the context of the performance assessment, the standard deviation can be interpreted as how much the performance of an infrastructure varies over the time;
- **Coefficient of Variation (or Relative Standard Deviation) over last month**: it is computed as the ratio between the standard deviation and the mean [26]. It has the same usage of the standard deviation, but, since it is dimensionless and can be expressed in percentage, it can be used as a standardized and easier to interpret measure of the performance variation;
- **Approximate Entropy over last month**: it is a measure of the "entropy" in a time-series [27]. It can be used to quantify the amount of regularity and the unpredictability of fluctuations in a time-series data. In the Benchmarking Suite, it is used (like the standard deviation) to measure how stable are the measurements of the metric over the time.

Each of these aggregated metrics is computed for each application-level metric (plus the *PerformanceIndex* metric) for each workload and for each infrastructure.

The only two metrics that are, at the moment, **shared with the Recommender are the mean and the coefficient of variation of the *PerformanceIndex***. In fact, these two metrics are the most synthetic indicators of respectively the mean overall performance of a workload in a given infrastructure and its stability over the last month. The Recommender algorithms use these values to take decisions on the best infrastructure (performance-wise) for a given application. All the other metrics are accessible from the *Visualization* component and from the Benchmarking Suite APIs.

The time period of one month chosen for the aggregation can be customized and experimentations with other time periods are ongoing to see if there are major differences and/or benefits using different time periods.

3.4 Infrastructures

The most important features of the Benchmarking Suite are the complete automation of the test executions and the abstraction over the target infrastructure technologies. To enable this, the Benchmarking Suite provides **extension points and multiple implementations of the *Executor* component**, that can be selected at runtime to match the required target infrastructure type. Table 5 presents the current implementation of executors and the supported infrastructures for each of them.

Table 5: Executors and supported infrastructure types

Executor	Description	Supported Infrastructures
Cloud	This executor uses the Cloud APIs to create VMs (with the possibility to select different flavour), set IPs and security groups, connect to the VMs (in SSH) and execute the tests.	Openstack Amazon Web Services
VMWare Cloud	Similar to the Cloud executor, but specific for the VMWare APIs.	VMWare Cloud
Kubernetes	This executor uses the Kubernetes API Server to create Kubernetes jobs that will run containers where the tests are executed. It also supports creation of deployment and services resources to run the needed services (e.g. databases) in separate containers. The executor can also select the node where the test will be executed.	Kubernetes KubeEdge
Docker	This executor uses the Docker Engine APIs to start containers where the services and the benchmarking tools are executed in containers. This executor only supports single node Docker clusters at the moment.	Docker Engine single node
Existing Server or VM (SSH)	This executor connects to an existing server or VM using the SSH protocol. Once connected, the Benchmarking Suite executes the tests running Unix Bash commands.	Any Unix machine
Local Execution	This executor executes the tests executing Unix Bash commands directly on the machine where it is running.	Any Unix machine

The Kubernetes and Docker executors have been developed in the Pledger project to support execution of tests in the ENG, i2CAT and FILL testbeds. The pre-existing Openstack and Existing Server executors have been updated and adapted in Pledger to support the HOLO testbed. Finally, the Local Execution executor has been used in Pledger in the learning process of the AppProfiler.

The FILL, i2CAT, ICCS and HOLO partners supported the development of the executors by providing requirements, configuring their testbeds and participating in the testing activities.

All the executors support the creation and management of network tunnels to be able to connect to the target infrastructures also in case of network firewall and/or VPN connections. This mechanism was implemented to be able to use Pledger testbeds that uses VPN connections for inter-testbed communication.

4 Installation and usage

4.1 Source code repository

The Benchmarking Suite source code is split in multiple Git repositories that are publicly available at https://gitlab.com/pledger/public/benchmarking/.

4.2 Licenses

Benchmarking Suite is released under the Apache 2.0 license. A full text of the license can be found at https://www.apache.org/licenses/LICENSE-2.0.

The Benchmarking Suite relies on external tools that are installed and executed on the target infrastructures. The licenses for the external tools are listed in section 3.2.

4.3 Software Requirements

The following requirements must be satisfied to install the Benchmarking Suite:

- Kubernetes cluster v1.15+ [28]with two persistent volumes for MongoDB and InfluxDB databases;
- Helm v3+ [29];
- (Optional) a Keycloak [9] instance configured with a realm, clients and users. If not available, it can be installed automatically along with the Benchmarking Suite;
- (Optional) an Apache Kafka [30] instance that will be used to send and receive model updates and send metric reports.

4.4 Installation

A Helm chart has been developed to automate the deployment of the Benchmarking Suite. It can be used as follows:

```
git clone https://gitlab.com/pledger/public/benchmarking/benchsuite-helm
cd benchsuite-helm
helm install bes1 . --namespace benchmarking --values custom-values.yaml
```

The source code of the chart, the default values, the possible custom values and the documentation are available at https://gitlab.com/pledger/public/benchmarking/benchsuite-helm.

4.5 Usage instructions

The documentation for the usage of the Benchmarking Suite both from GUI and from Command Line Interface (CLI) is available at https://benchmarking-suite.readthedocs.io/.

In addition, two video tutorials are available in the Pledger project's YouTube channel[3] that describe how to use the benchmarking tool in the Pledger system:

- https://www.youtube.com/watch?v=1R6QYYjn6OM
- https://www.youtube.com/watch?v=oJGZCa8SUUI

[3] https://www.youtube.com/channel/UCXV6V9rJ0ZvWhXeoWvDsArQ

4.6 Target Infrastructures Requirements and Costs

The Benchmarking Suite is meant to automatically connect to target infrastructures and execute code to assess their performance. There are few requirements that target infrastructures must satisfy in order to be used by the Benchmarking Suite:

- the target infrastructure must provide a **programmatic access** (e.g., an API) that the Benchmarking Suite can use to automate the management of resources and execution of code on the infrastructure. The Benchmarking Suite already supports multiple technologies (i.e., Docker, Kubernetes, Openstack, VMWare, AWS, Unix VMs) and other can be implemented thanks to specific extension points;
- **network connectivity** must be ensured between the Benchmarking Suite instance and the target remote infrastructure endpoint. Since there are scenarios where infrastructures are not publicly exposed to internet, the Benchmarking Suite implements some mechanisms (e.g. network tunnels) to try to overcome issues created by network firewalls and VPN connections (section 3.4);
- the Benchmarking Suite needs valid **credentials** to authenticate to the target infrastructure. The credentials must also be associated with the required **permissions** needed to execute the tests on the target infrastructure. The required permissions change from technology to technology (e.g., create VMs on Cloud infrastructures, create jobs in Kubernetes clusters, run user-space scripts in servers). The Benchmarking Suite **ensures the security of credentials** both at rest (through encryption in the database) and in transit (using HTTPS for all communications).

One aspect not directly managed by the Benchmarking Suite at the moment is **the cost of executing benchmark tests** on the target infrastructures. In fact, the Benchmarking Suite uses the infrastructure on behalf of the user (creating and managing resources and running workloads) using her credentials. If these operations have a cost, **the user will incur in the costs associated with the execution of the tests**. The Benchmarking Suite does not take into account these costs and there is not a modelling of infrastructure costs. However, the **Benchmarking Suite put in places some actions to optimize the usage of resources**: a) clustering of test executions in order to minimize creation of new resources (e.g., reuse same VMs or services for multiple tests), b) a hard limit (of one hour) for the maximum execution time for a test (this also act as a safeguard mechanism for case of hanging executions). In addition, other actions that the user can take to minimize the costs are: a) use, when available, public performance results, b) set quotas for infrastructure usage in order to not incur in unexpected costs, c) adjust the execution times and frequency and d) limit the set of tests to be executed only to the ones really necessary.

A possible future work to improve the awareness of costs in the Benchmarking Suite would be the introduction of a cost models for the infrastructure (to be able to predict the costs associated to the execution of the tests) and the automatic tuning of tests to execute and the execution time and frequency based on the previous test results and the application profiles the user is interested in.

5 Conclusions

This document accompanies the final release of the components in the benchmarking subsystem developed in the context of the project's Task "T3.1 - Performance Measurements and Classification". The document is an update of document "D3.1 Performance Measurements and classification tools I" [1] delivered at M18 that described the work in T3.1 during the first period of the project.

The document reported the main functional and technical aspects of the prototype and the guidelines to use it, **focusing on the progresses and the work done in the second project period**. In particular, the main achievements in this period were: a) the development of Docker executor, b) the integration of new benchmarking tools, c) the definition of higher level metrics and the development of the Analytics component, the d) implementation of mechanisms to measure, store and visualize system-level metrics and e) minor improvements of multiple other aspects and components (e.g., visualization dashboards, integrated tools and workloads, execution results parsing and metrics computation).

The Benchmarking Suite is released under the open-source license Apache version 2.0 and the source code is available at https://gitlab.com/pledger/public/benchmarking/. The tool also provides an online user documentation available at https://benchmarking-suite.readthedocs.io/.

This release represents **the final release of the Benchmarking Suite prototype** and no further development activities are expected in task T3.1. However, activities related to this prototype will continue in the project in WP5 where the integration with the other Pledger components and testbeds will be set-up, tested and tuned. In addition, in WP5, the prototype will be used to assess performance of the project testbeds in the context of the use case executions. Finally, the approach proposed in the Pledger project of using benchmarking to predict the performance of use cases applications on different infrastructures will be validated by two project's KPIs: a) KPI#6 "Improved experienced performance and stability of virtualised resources through enhanced provider resource management" that will validate how the predictions done by benchmarking will results in a better performance of the deployed application and b) KPI #12 "Accuracy of classification of Use Case applications to benchmark categories" that will validate how well benchmarking tests can be profiled and matched with real applications.

Future research directions have been identified a) to improve the performance assessments by using system-level metrics already collected during the executions and b) to optimize the resource usage during the execution of tests by analysing previous executions and modelling infrastructural costs.

Appendix I - Benchmarking Suite APIs

In order to retrieve metrics in the *Metrics DB*, an API is available that accepts queries in the *InfluxDB InfluxQL* language [31]. The API is exposed by the *InfluxProxy* component at the "/query" path. The query text must be included in the "q" parameter and a valid Keycloak token must be present in the "Authorization" HTTP header.

An example of a call to get results data is:

```
curl '<INFLUX_PROXY_BASEURL>/query' --data-urlencode "db=benchsuite" --data-urlencode "q=SELECT mean(\"duration_s\") FROM \"SysBench\" WHERE (\"provider\" = \"testprovider\" AND time >= now() - 3d)"
```

In order to access Benchmarking Suite configuration objects a REST API is available. The endpoint is exposed by the *API* component and all the available operations are listed in Table 6. The calls are authenticated and needs an "Authorization" header in the HTTP request containing a valid token issued by Keycloak. All responses return JSON objects.

An example of a call to the REST API is:

```
curl -H 'Accept: application/json' -H "Authorization: Bearer <TOKEN>" '<API_BASEURL>/api/v3/providers'
```

Table 6: Benchmarking Suite REST API

Method	URI	Description
GET	/executions/	Returns the list of executions
GET	/executions/user_actions	Returns the list of available actions on the executions for the logged in user
GET	/executions/{execution_id}	Returns the execution with the given id
GET	/executions/{execution_id}/request	Returns the schedule that generated the execution with the given id
GET	/executions/{execution_id}/runs	Returns the logs and results for the execution with the given id
POST	/organizations/	Creates a new organization
GET	/organizations/	Returns the list of organizations
DELETE	/organizations/{organization_id}	Deletes the organization with the given id
GET	/organizations/{organization_id}	Returns the organization with the given id
PUT	/organizations/{organization_id}	Update the organization with the given id
POST	/organizations/{organization_id}/users	Associates a user to the organization with the given id
GET	/organizations/{organization_id}/users	Returns the list of users belonging to the organization with the given id
DELETE	/organizations/{organization_id}/users/{user_id}	Deletes the user with the given user id from the organization with the given organization id
POST	/providers/	Creates a new provider

Method	URI	Description
GET	/providers/	Returns the list of providers
GET	/providers/types	Returns the list of supported provider types
GET	/providers/user_actions	Returns the list of available actions on the providers for the logged in user
PATCH	/providers/{provider_id}	Updates the provider with the given id only for the provided values
DELETE	/providers/{provider_id}	Deletes the provider with the given id
GET	/providers/{provider_id}	Returns the provider with the given id
PUT	/providers/{provider_id}	Updates the provider with the given id overriding the current version
GET	/providers/{provider_id}/check	Tests the configuration of the provider with the given id trying to connect to it
GET	/providers/{provider_id}/user_actions	Returns the list of available actions on providers for the logged-in user
GET	/providers/{provider_id}/vm/flavours	For Cloud providers, returns the list of available VM sizes
GET	/providers/{provider_id}/vm/images	For Cloud providers, returns the list of available VM images
GET	/providers/{provider_id}/vm/params	For Cloud providers, returns the list of configurable VM parameters
GET	/providers/{provider_id}/vm_images	For Cloud providers, returns the list of available VM images
GET	/providers/{provider_id}/vm_sizes	For Cloud providers, returns the list of available VM sizes
POST	/schedules/	Creates a new schedule
GET	/schedules/	Returns the list of schedules
GET	/schedules/user_actions	Returns the list of available actions on schedules for the logged-in user
DELETE	/schedules/{schedule_id}	Deletes the schedule with the given id
GET	/schedules/{schedule_id}	Returns the schedule with the given id
PUT	/schedules/{schedule_id}	Updates the schedule with the given id
GET	/schedules/{schedule_id}/user_actions	Returns the list of available actions on the schedule with the given id for the logged-in user
POST	/users/	Creates a new user
GET	/users/	Returns the list of users
DELETE	/users/{user_id}	Deletes the user with the given id
GET	/users/{user_id}	Returns the user with the given id
PUT	/users/{user_id}	Updates the user with the given id
GET	/users/{username}/scopes	Returns the scopes for the user with the given username
POST	/workloads/	Creates a new workload

Method	URI	Description
GET	/workloads/	Returns the list of workloads
GET	/workloads/download	Returns the list of workloads
GET	/workloads/user_actions	Returns the list of actions available on workloads for the logged-in user
DELETE	/workloads/{workload_id}	Deletes the workload with the given id
GET	/workloads/{workload_id}	Returns the workload with the given id
PUT	/workloads/{workload_id}	Updates the workload with the given id
GET	/workloads/{workload_id}/download	Returns the workload with the given id
GET	/workloads/{workload_id}/user_actions	Returns the list of actions on the workload with the given id for the logged-in user

Appendix II – Benchmarking Suite Demonstration

This section presents the basic flow of the Benchmarking Suite usage along with some screenshots:

1. The user logs-in in the web-based Benchmarking Suite GUI. This process involves the redirect to Keycloak for the authentication (Figure 6).
2. The user creates a new Provider: in case of Kubernetes, she will be asked to insert the API Server endpoint, user token and namespace (Figure 7).
3. Alternatively, the user can register a new infrastructure in the Pledger Configuration service. The Benchmarking Suite will be synchronised automatically, and a corresponding new Provider will be created (Figure 8).
4. The user creates a new Schedule (Figure 9): they select a) the Provider created in the previous step, b) one benchmark test (e.g., SysBench) a repetition frequency (e.g., 1 hour) and the visibility of results (e.g., private). After saving, the system will schedule the execution of the new test (can require up to 1 minute).
5. The user can check the execution of the Schedule created in the previous step in the Executions page. They can follow the status of execution (i.e., Pending, Running, OK, Error) and, when it is completed, they can also check the logs and the metrics collected (Figure 10).
6. The user can analyse the results for all their executions and providers, as well as all public executions, in the Results page. The page presents an interactive set of charts where metrics are aggregated by time, provider, and test type (Figure 11, Figure 12 and Figure 13).
7. Aggregated results are also visible in the Pledger Recommender component that uses them as input to take decisions (Figure 14).

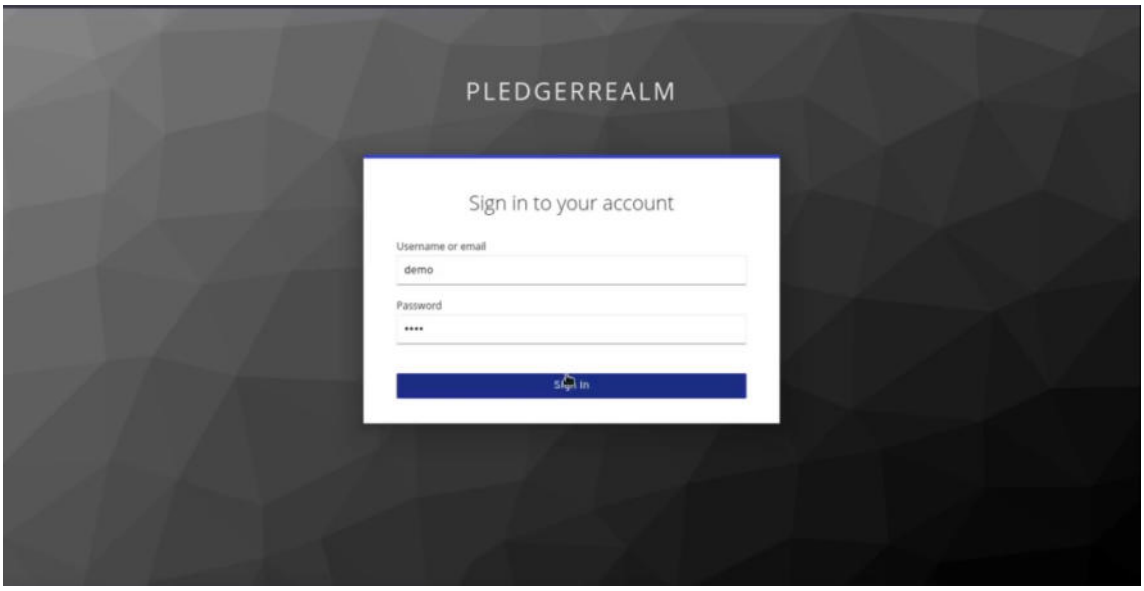

Figure 6: Benchmarking Suite log-in page

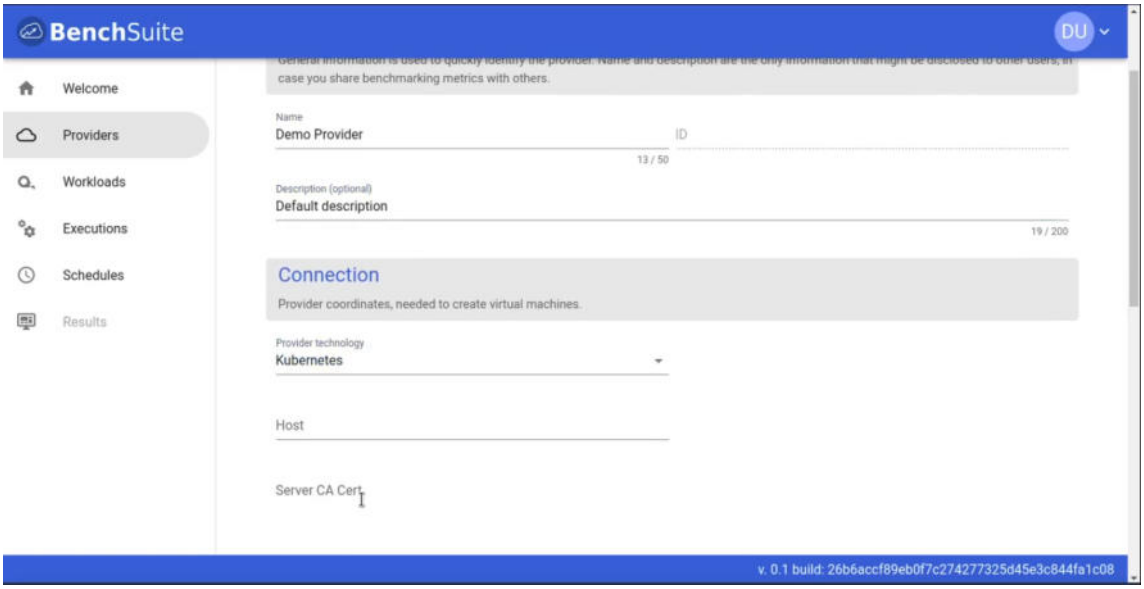

Figure 7: Benchmarking Suite Provider creation page

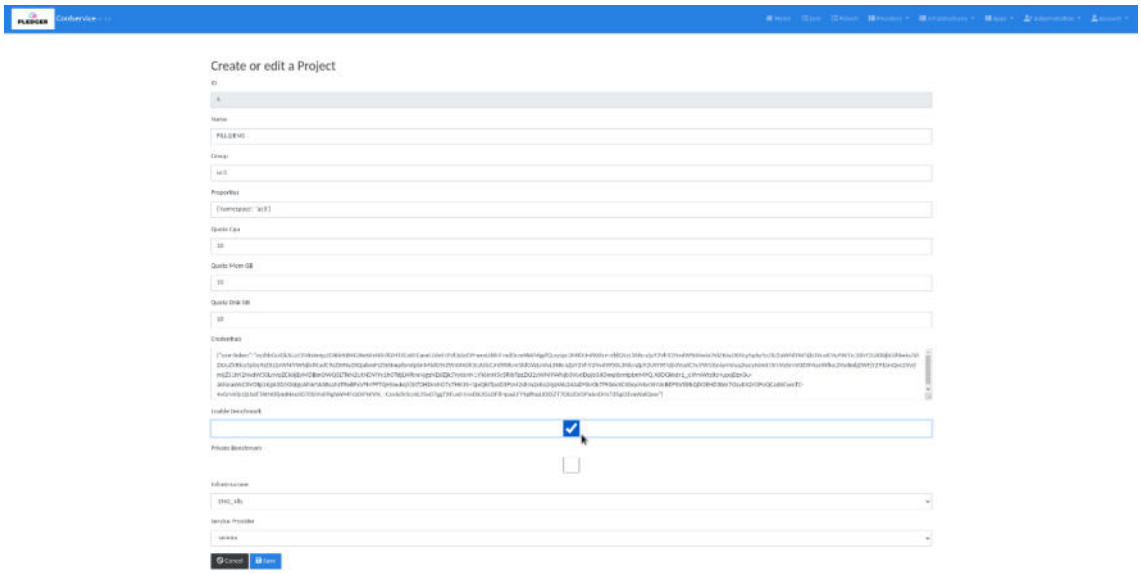

Figure 8: Configuration Service Project creation

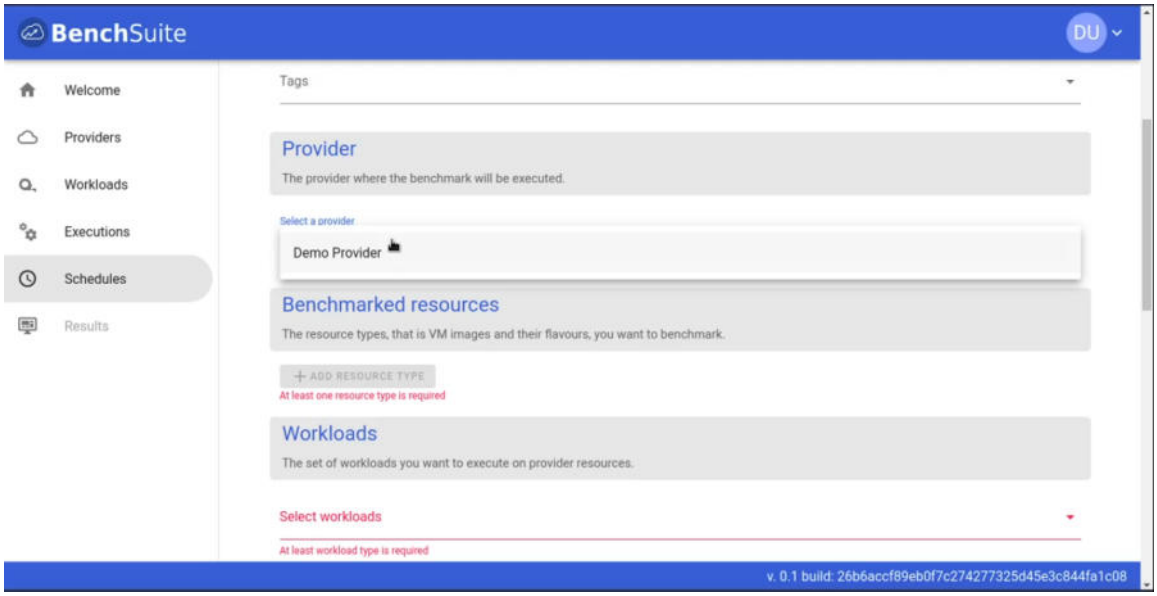

Figure 9: Benchmarking Suite Schedule creation page

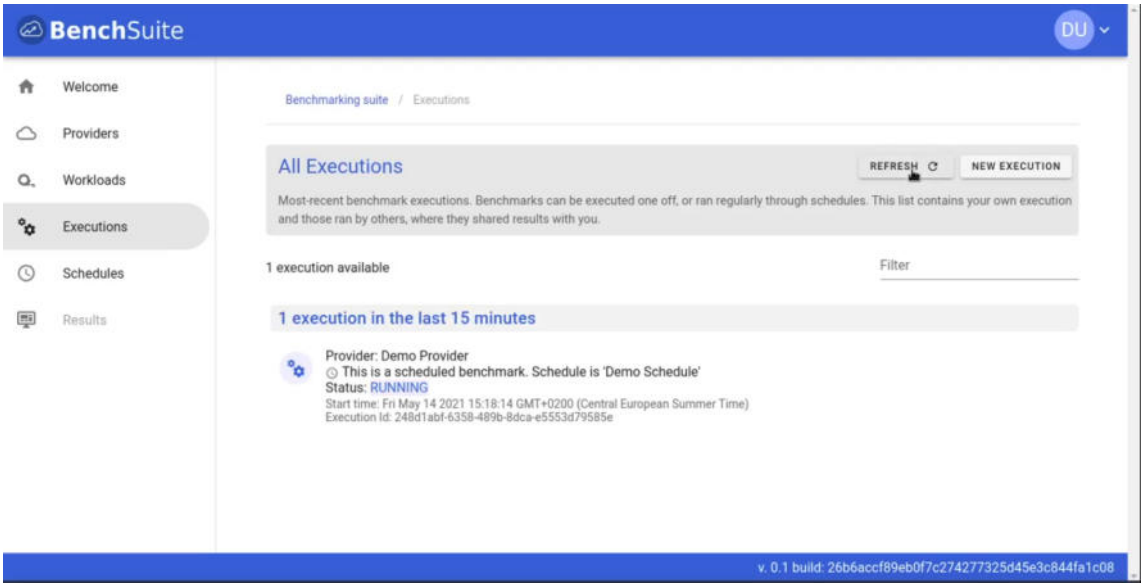

Figure 10: Benchmarking Suite Executions page

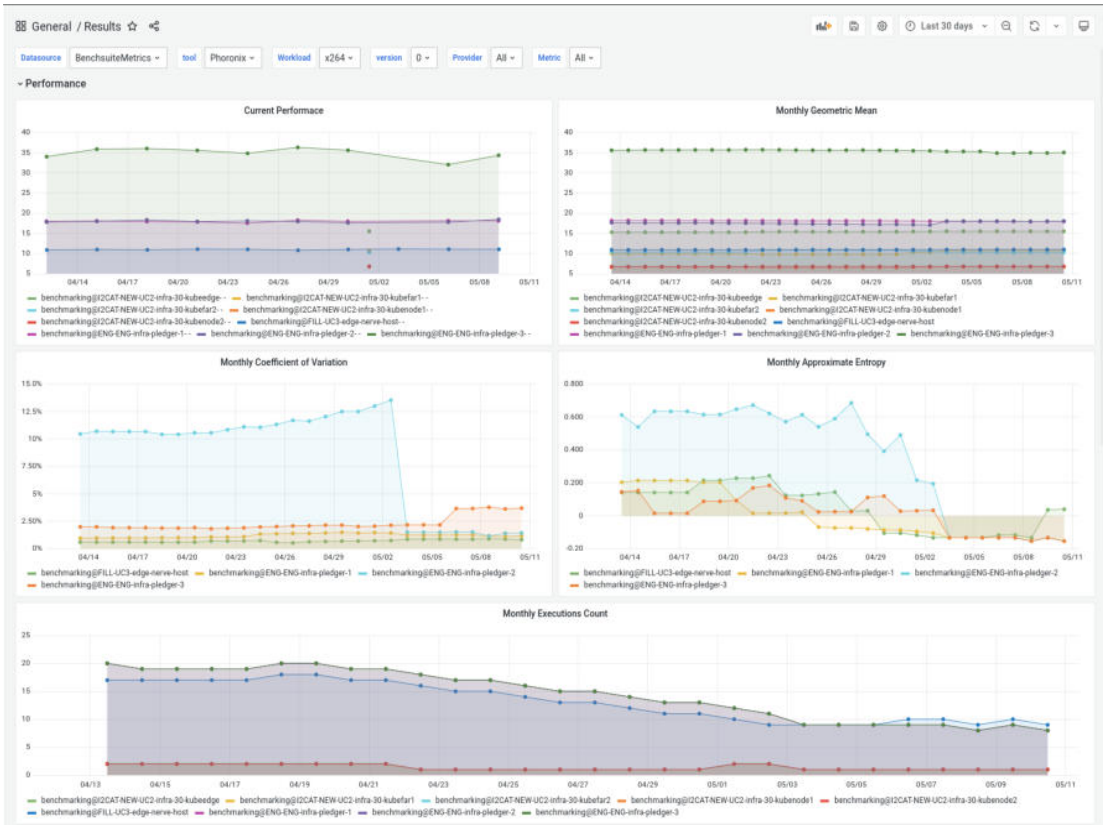

Figure 11: Visualization of PerformanceIndex metric

Figure 12: Visualization of application-level metrics

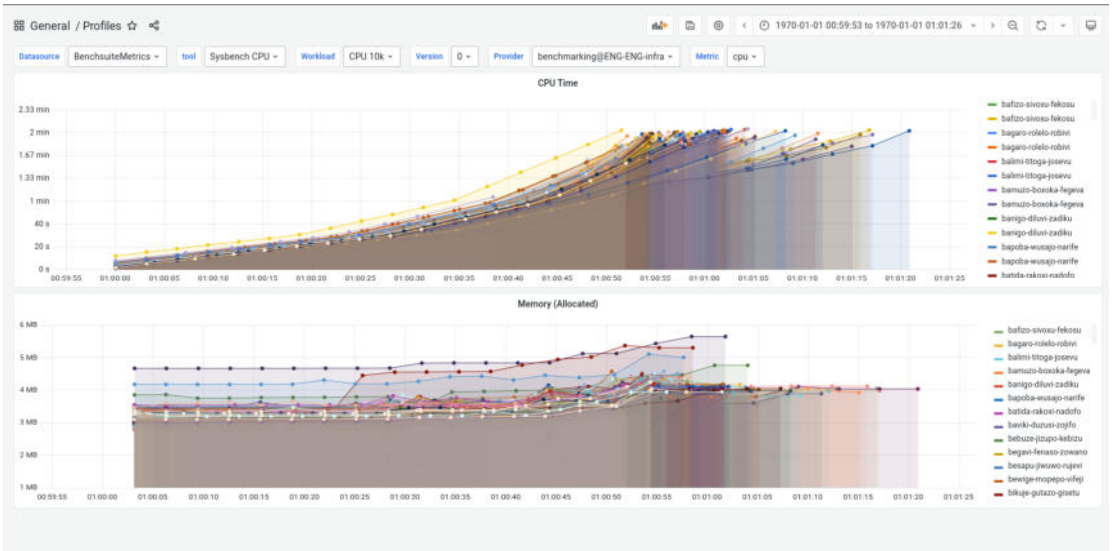

Figure 13: Visualization of system-level metrics

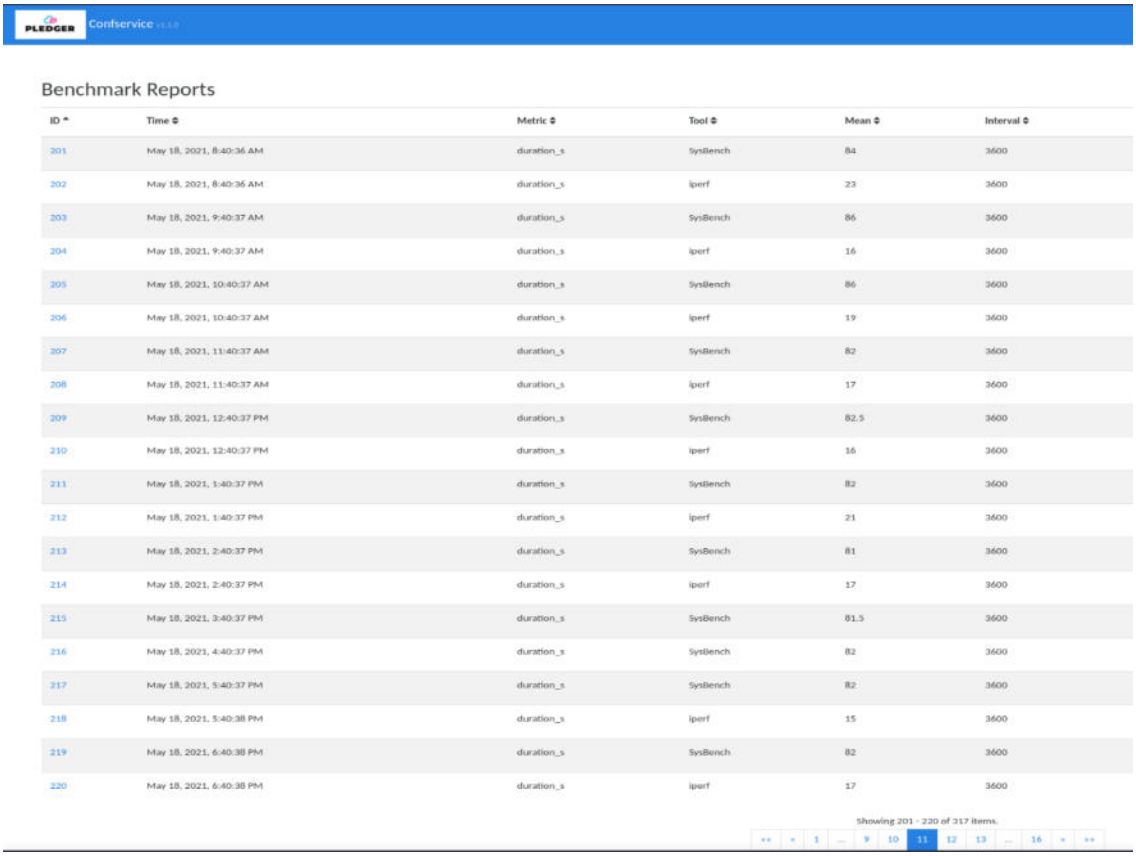

Figure 14: Benchmarking reports sent to the Configuration service

References

[1] PLEDGER. D3.1 - Performance Measurements and classification tools I. Gabriele Giammatteo. 2020. http://pledger-project.eu/content/deliverables. Retrieved 2022-05-31.

[2] PLEDGER. D2.2 - Pledger Requirements Analysis, Iadanza, Francesco. 2020. http://pledger-project.eu/content/deliverables. Retrieved 2022-05-31.

[3] PLEDGER. D2.3 - Pledger Overall Architecture. Voutyras, Orfefs. 2020. http://pledger-project.eu/content/deliverables. Retrieved 2022-05-31.

[4] CloudPerfect EU research project. https://cordis.europa.eu/project/id/732258, retrieved 2021-10-06.

[5] MongoDB home page. https://www.mongodb.com, retrieved 2021-05-27.

[6] InfluxDB home page. https://www.influxdata.com/products/influxdb/, retrieved 2021-10-06.

[7] Prometheus home page. https://prometheus.io/, retrieved 2021-10-06.

[8] Vue.js home page. https://vuejs.org/, retrieved 2021-05-27.

[9] Keycloak home page. https://www.keycloak.org, retrieved 2021-05-27.

[10] Grafana home page. https://grafana.com, retrieved 2021-05-27.

[11] Center for Internet Security – Docker. https://www.cisecurity.org/benchmark/docker, retrieved 2022-05-10

[12] Center for Internet Security – Kubernetes. https://www.cisecurity.org/benchmark/kubernetes, retrieved 2022-05-10

[13] CloudSuite In Memory Analytics home page. https://github.com/parsa-epfl/cloudsuite/blob/main/docs/benchmarks/in-memory-analytics.md, retrieved 2022-05-10

[14] DaCapo test suite. http://dacapo-bench.org/, retrieved 2021-10-06.

[15] FileBench test suite. https://github.com/filebench/filebench, retrieved 2021-10-06.

[16] Iperf tool. https://iperf.fr/, retrieved 2021-10-06.

[17] RabbitMQ Performance Testing Tool home page. https://rabbitmq.github.io/rabbitmq-perf-test/stable/htmlsingle/, retrieved 2022-05-11

[18] Phoronix Testsuite. https://www.phoronix-test-suite.com/, retrieved 2021-10-06.

[19] SciKit Learn homepage. https://scikit-learn.org/, retrieved 2022-05-11

[20] Sysbench tool. https://github.com/akopytov/sysbench, retrieved 2021-10-06.

[21] YCSB tool. https://github.com/brianfrankcooper/YCSB, retrieved 2021-10-06.

[22] Tensorflow Benchmarks repository. https://github.com/tensorflow/benchmarks/, retrieved 2022-05-11

[23] WebFrameworks. https://github.com/the-benchmarker/web-frameworks, retrieved 2021-10-06

[24] Benchmarking Suite documentation - Benchmarks and Metrics. https://benchmarking-suite.readthedocs.io/en/latest/benchmarks.html, retrieved on 2021-10-06

[25] Docker home page. https://www.docker.com/, retrieved 2021-05-27.

[26] Everitt, Brian. The Cambridge Dictionary of Statistics. Cambridge, UK New York: Cambridge University Press, 1991.

[27] Pincus SM. Approximate entropy as a measure of system complexity. Proc Natl Acad Sci U S A, 1991.https://kubernetes.io/https://grafana.com/https://docs.influxdata.com/influxdb/v1.8/query_language/https://apache.kafka.org/https://www.python.org/https://golang.org/

[28] Kubernetes home page. https://kubernetes.io, retrieved 2021-05-27.

[29] HELM home page. https://helm.sh, retrieved 2021-05-27.

[30] Kafka home page. https://kafka.apache.org/, retrieved 2021-05-27.

[31] Influx Query Language (InfluxQL). https://docs.influxdata.com/influxdb/v1.8/query_language/, retrieved 21-05-27.