D3.1 Performance Measurements and classification tools I

Document Identification			
Status	Final	**Due Date**	31/05/2021
Version	1.0	**Submission Date**	01/06/2021

Related WP	WP3	**Document Reference**	D3.1
Related Deliverable(s)	D2.2, D2.3	**Dissemination Level (*)**	PU
Lead Participant	ENG	**Lead Author**	Gabriele Giammatteo
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

Document History			
Version	Date	Change editors	Changes
0.1	01/03/2021	Gabriele Giammatteo (ENG)	Table of Contents circulated
0.2	23/04/2021	Gabriele Giammatteo (ENG)	Sections 2, 3 and 4 provided
0.3	05/05/2021	Francesco Iadanza (ENG)	Section 5 provided
0.4	14/05/2021	Gabriele Giammatteo (ENG)	Final version ready for internal review
0.5	17/05/2021	Orfeas Voutyras (ICCS)	Internal review
0.6	19/05/2021	Roi Sucasas (ATOS)	Internal review
0.7	20/05/2021	Gabriele Giammatteo (ENG)	Review comments merged
0.8	27/05/2021	Carmen San Roman (ATOS)	Quality assurance review
1.0	31/05/2021	Lara López (ATOS)	Version ready for submission

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Gabriele Giammatteo (ENG)	20/05/2021
Quality manager	Carmen San Román (ATOS)	27/05/2021
Project Coordinator	Lara López (ATOS)	31/05/2021

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
API	Application Programming Interface
CLI	Command Line Interface
CRUD	Create, Read, Update and Delete
Dx.y	Deliverable number y belonging to WP x
DSS	Decision Support System
FC	Functional Component
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
Mx	Project Month x
MVP	Minimum Viable Product
OS	Operating System
REST	REpresentational State Transfer
Tx.y	Task number y belonging to WP x
WP	Work Package

Executive Summary

This document accompanies the delivery of the prototype developed in Task 3.1 (T3.1) "Performance Measurements and Classification" for the Pledger's Benchmarking functional subsystem. The main responsibility of the Benchmarking subsystem is to provide performance data for configured infrastructures to better characterise them and to optimise the orchestration and deployment decisions for the applications. The prototype released is based on the asset "Benchmarking Suite" brought and developed in the project by Engineering Ingegneria Informatica (ENG).

The Benchmarking Suite is a cloud native, Kubernetes-based, microservice oriented application that is constituted by multiple containers communicating over network. The Benchmarking Suite is able to scale-in/out its components by talking 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 functionality. 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 service by a) listening to changes in the infrastructures managed by the system to automatically schedule/ update execution of benchmarks on those infrastructures and b) by sending performance evaluation results to the Configuration service (that will make them available to the other Pledger components).

The Benchmarking suite is an open-source software available at https://gitlab.res.eng.it/benchsuite and released under the Apache 2.0 License. The application also includes a Helm chart to easily deploy it in any Kubernetes cluster.

During the first development iteration, all expected requirements have been realized, leaving two main functional requirements (the user provided benchmarks and customizable dashboards) for the second development iteration.

For demonstration purposes, two scenarios have been prepared, one in which the Benchmarking Suite is used as stand-alone tool, and another one which demonstrates the integration with the Configuration Service.

1 Introduction

1.1 Purpose of the document

This document accompanies the 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.

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 Decision Support System (DSS) 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 6 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 guide of the tools.
- **Section 5** presents the demonstration of the tools with the description of the validation scenarios.
- **Section 6** presents the conclusions and next steps for this first iteration of the tools.

1.4 Glossary adopted in this document

- **Pledger core system**. The system that will be the "minimum viable product (MVP)" of Pledger, the system that will be freely available to users and other projects and initiatives. It is the exploitable part of the project and it contains 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" [1], which acts as the roadmap for all development and integration tasks of the project.

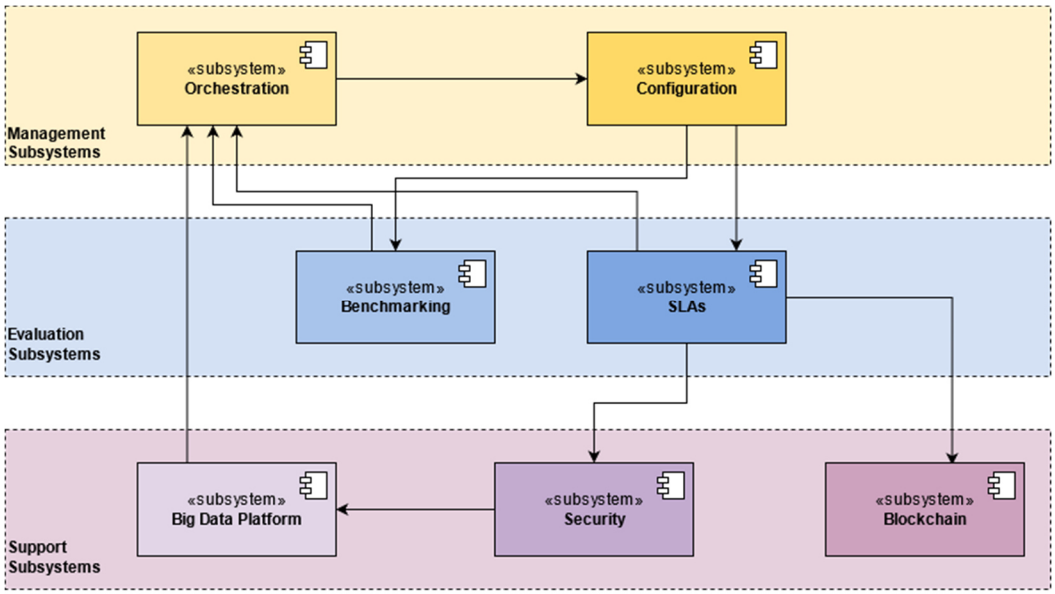

Figure 1: Pledger's Core Subsystems

The main responsibility of the Benchmarking subsystem is to provide performance data for configured infrastructures to better characterise them and to optimise the deployment and orchestration of applications.

The benchmarking process in Pledger is designed to be as automated as possible, with auto-discovery of infrastructure resources, automated scheduling and execution of benchmarking tests, aggregation and sharing of results, etc. This is made possible thanks to the integration with the Configuration subsystem a) to read the infrastructure and benchmarking configurations and b) to share infrastructure's performance data as results of the benchmarking activities.

Figure 2 shows the components that compose the Benchmarking subsystem with their connections to other components, while Table 1 summarizes the main functionality for each component.

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 **Benchmarking Library**. Beside the automatic settings, the user can adjust and customize the behaviour of benchmarking from and ad-hoc GUI (the **Benchmarking Creator** component) that can be used to activate/deactivate default tests, define new tests, set frequency and visibility of results.

In order to execute the tests to evaluate infrastructure performance, the **Benchmarking Executor** connects to the external infrastructures using credentials available in the Configuration subsystem and executes tests 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 Benchmarking GUI. In addition, a subset of metrics is shared with the Configuration subsystem to make them available to the other Pledger components (e.g., Recommender).

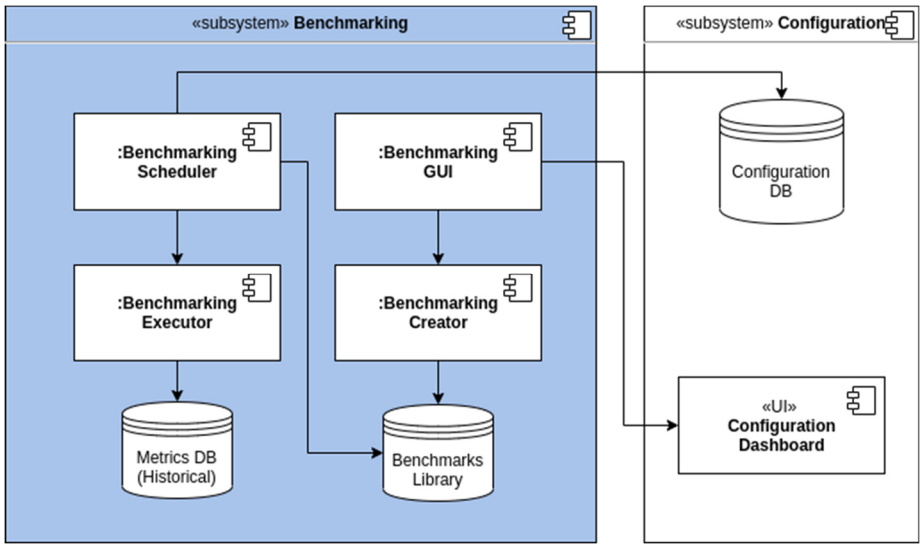

Figure 2: Benchmarking subsystem components

Table 1: Benchmarking subsystem Functional Components (FC)

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 executing 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.

3 Technical description

This section presents the main technical aspects of the implementation of the Pledger's Benchmarking subsystem. All the functional components of the Benchmarking subsystem have been implemented adapting and evolving a technical asset called **Benchmarking Suite** brought and developed in the project by Engineering Ingegneria Informatica (ENG).

3.1 Baseline technologies and dependencies

Table 2 lists the main technologies and libraries on top of which the Benchmarking Suite is based.

Table 2: Benchmarking Suite baseline technologies and dependencies

Name	Description	Version
Docker [2]	Container engine used to build, package and run containerized applications. In the Benchmarking Suite, the Docker Image format is used to package both the application components and the benchmark tests.	>= 18.09
Kubernetes [3]	Kubernetes is container orchestration system for automating computer application deployment, scaling, and management. In the Benchmarking Suite, it is used as platform for creating and running the benchmark executors and for running the application itself.	>= 1.15
Keycloak [4]	Identity and Access Management system that supports OAuth 2.0 protocol. In the Benchmarking Suite, it is used for storing users and permissions, log-in and single-sign-on.	>= 12.0
MongoDB [5]	MongoDB is an open-source NoSQL database that uses JavaScript Object Notation (JSON)-like documents with optional schemas. In the Benchmarking Suite, MongoDB is used to store the application model.	>= 4.2.8
Grafana [6]	Grafana is a data analytics and interactive visualization web-based application with charts, graphs and alerts. In the Benchmarking Suite, it is used to visualize the benchmarking results.	>= 7.5.2
InfluxDB [7]	InfluxDB is a fast and efficient time-series database. In the Benchmarking Suite, it is used to store metrics collected during benchmarking tests.	< 2.0
Kafka [8]	Kafka is a software bus with streaming processing developed by the Apache Software Foundation. In the Benchmarking Suite, it is used to exchange data and events with the other Pledger components.	>= 2.0
Python [9]	Python is a general-purpose, dynamically-typed programming language. In the Benchmarking Suite, it is used to program the majority of the application components.	>= 3.8
Golang [10]	Go is a statically typed, compiled programming language designed at Google. In the Benchmarking Suite, it is used to realize the InfluxProxy component.	>= 1.14
Vue.js [11]	Vue.js is a progressive, reactive front-end JavaScript framework for building user interfaces and single-page applications. In the Benchmarking Suite, it is used to build the GUI component.	< 3.0

3.2 Components Architecture

The Benchmarking Suite is a cloud native, Kubernetes-based, microservice oriented application that is constituted by multiple containers communicating over network. The Benchmarking Suite is able to scale-in/out its components by talking with the underlying Kubernetes cluster to adapt to the application load.

Figure 3 provides the deployment schema of the main software components of the Benchmarking Suite (and the main connections between them) that realize the functionalities of the Benchmarking subsystem.

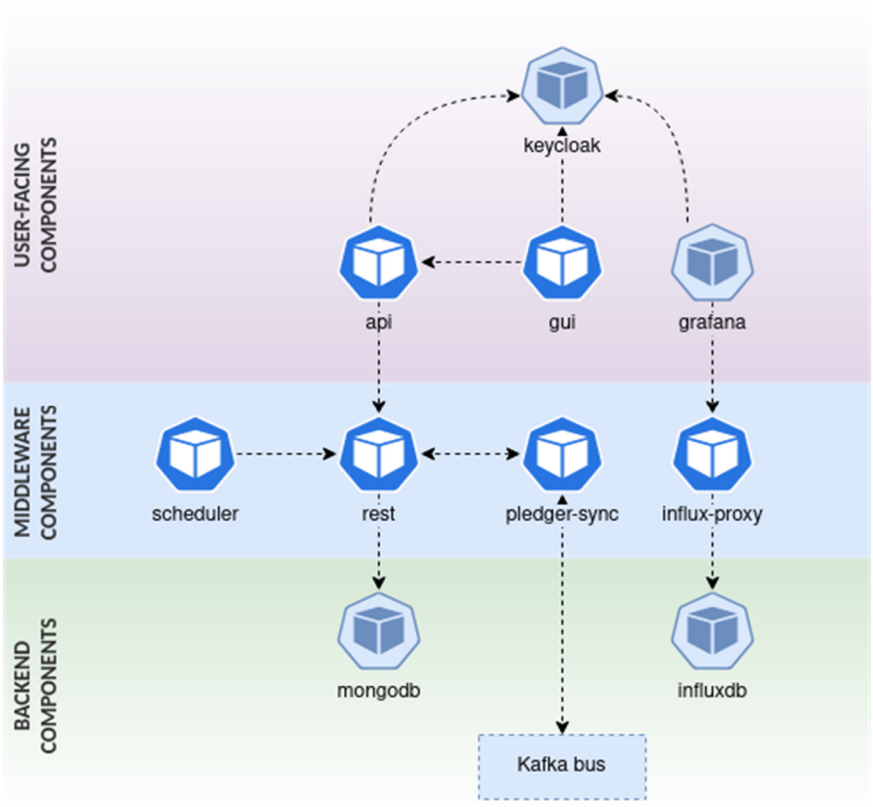

Figure 3: Benchmarking Suite component architecture

The back-end components consist of a **MongoDB** database that is used to store the model of the application (i.e., providers, benchmarks, schedules, executions) and an **InfluxDB** database to store metrics collected during the tests executions. InfluxDB has been chosen for storing metrics instead of MongoDB, since it is specific and highly optimized for time-series and provides more speed and querying capabilities.

The **PledgerSynchronizer** component provides a way (through the **Kafka** bus) to integrate with the other Pledger components by sharing the application model and the metrics as well as to receive updates of the infrastructure models.

Access to the application model for other components is guaranteed by the **Rest** component that provides Create, Read, Update and Delete (CRUD) operations for the model objects. Access to metrics, instead, goes through the **InfluxProxy** component which has the main objective of filtering and enriching queries to enforce the visibility (private, public, organization) of results.

The components exposed to the users are: the **API** that provides a programmatic access to the application model (similar to the Rest, but enforcing authentication and authorization of the requests), the **GUI** that consists in a web-based portal to manage all the application objects and a customized **Grafana** instance that provides visualization and analysis tools for the metrics. All the exposed components support OAuth 2.0 authentication and authorization implemented through a **Keycloak** instance.

Finally, the **Scheduler** component is responsible for the automation of benchmark execution. The scheduler monitors the benchmarking schedules defined and triggers the execution of each of them based on the schedule's frequency. As shown in Figure 4, it uses the **Kubernetes API server** (accessing with the **executor service account**) to create a Kubernetes job running a benchmarking executor (named **multiexec**) and configure it with the target infrastructure and benchmarking configurations (stored in Kubernetes secrets). The executor job connects to the target infrastructure, sets-up the environment, executes the tests, collects the results and cleans-up the environment. Before terminating, the executor stores the execution logs and metrics collected in the Benchmarking Suite backend. After the executor has terminated, the Scheduler deletes all the resources created for the execution.

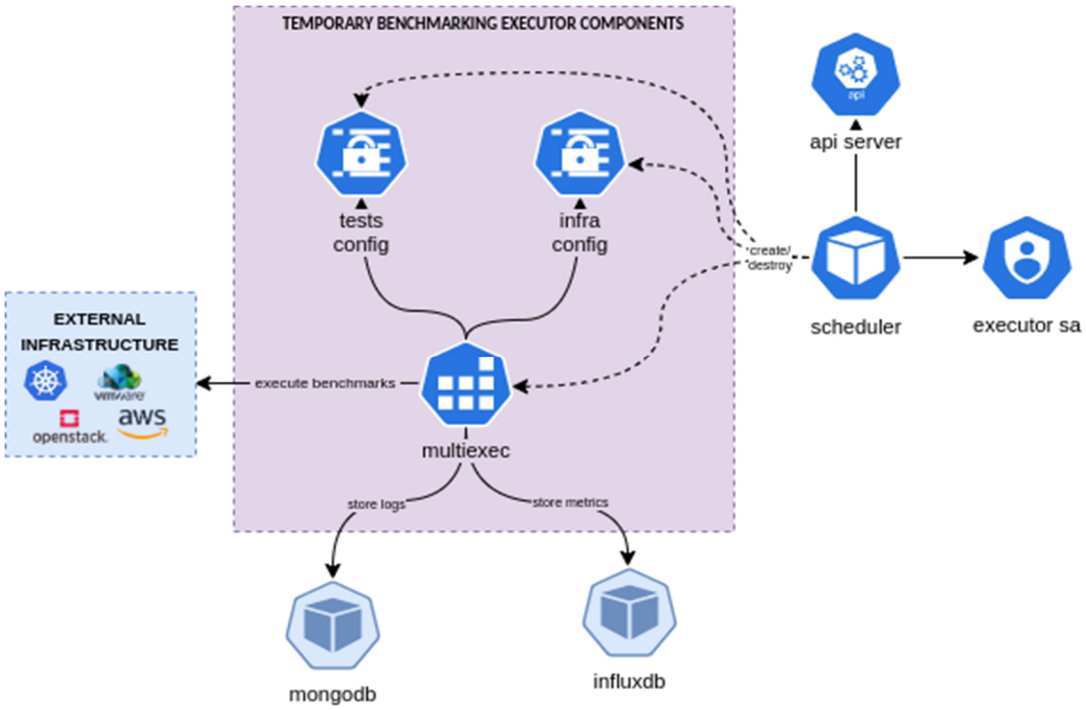

Figure 4: Benchmarking Suite tests execution workflow

The deployment of the Benchmarking Suite is completed by multiple Kubernetes secrets and configuration maps (shown in Figure 5) that are used by the application components to store credentials and configuration. Managing these elements in Kubernetes resources instead of embedding them in the containers makes it easier to update them without re-deploying the application and to share them between components.

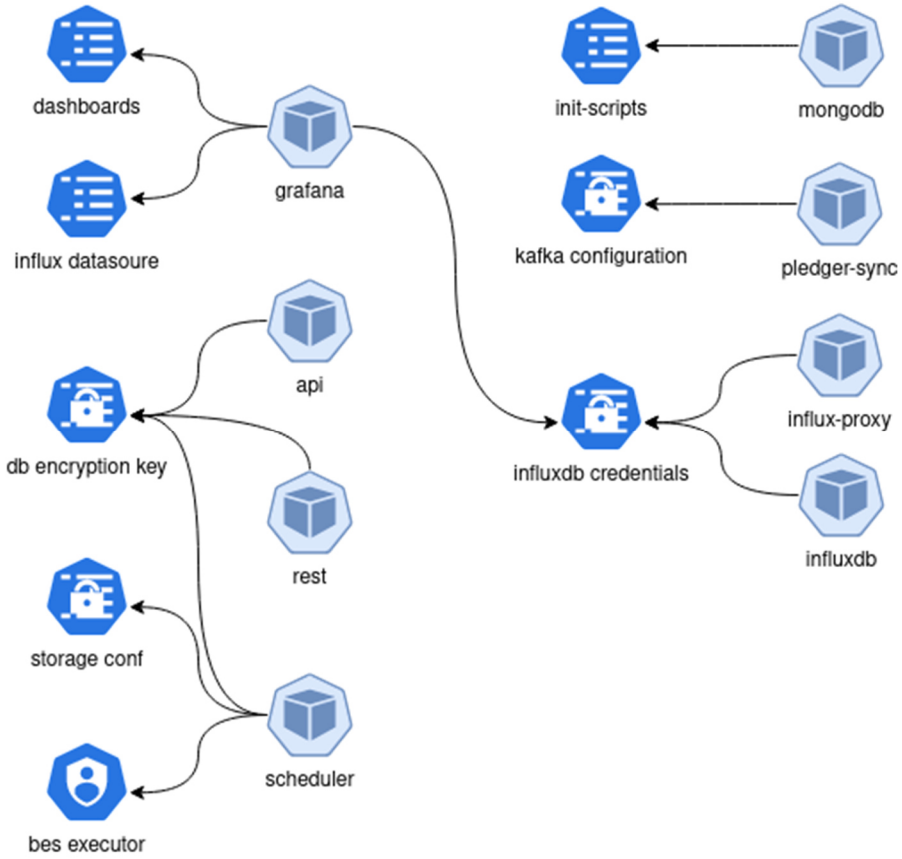

Figure 5: Benchmarking Suite configuration Kubernetes resources

3.3 Interfaces provided

The Benchmarking Suite provides different ways to interact with the application and the data produced:

- a web-based GUI, to manage the application model objects (see section 3.4) and results. Some screenshots of the GUI are presented in section 5.3;
- a REST API to access the application model objects programmatically. The endpoint is exposed by the API component and all the available operations are listed in Table 3. The calls are authenticated and needs an Authorization HTTP header containing a valid token issued by Keycloak to work. All responses return JSON objects;
- an API to query the benchmarking results data. It uses the InfluxDB InfluxQL language [7], and it is exposed by the InfluxProxy component at the */query* path. The query 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 the REST API is:

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

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)"
```

Table 3: 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
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

Method	URI	Description
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
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

3.4 Data models

In this section, the main concepts managed by the Benchmarking Suite and the way they are modelled in the application are presented.

3.4.1 Provider

Provider object represents the target infrastructure that needs to be benchmarked. It contains needed information to access the infrastructure (e.g., endpoint URL, credentials). Multiple technologies are currently supported: Openstack, VMWare, Amazon AWS, Azure and Kubernetes.

The code below shows two providers (one Openstack and one Kubernetes) as returned by the APIs. There is a set of common fields (i.e., id, name, description, driver, class, grants) plus a set of custom fields that depends on the provider technology.

```
//Openstack Provider

  {
    "id": "26940e91-360f-42b4-be49-1147e034cc8e",
    "name": "FIVicTpl",
    "driver": "openstack",
    "class": "benchsuite.stdlib.provider.libcloud.LibcloudComputeProvider",
    "description": "Default description",
    "grants": {
      "viewer": "org:public",
      "owner": "usr:gabriele"
    },
    "service_properties": {
      "post_create_script": {
        "centos": "sudo sed -i 's/requiretty/!requiretty/' /etc/sudoers",
        "debian": "sudo hostname localhost"
      }
    },
    "network": "node-int-net-01",
    "secret_key": "********",
    "access_id": "",
    "region": "Vicenza",
    "auth_url": "http://cloud.lab.fiware.org:4730/",
    "tenant": "ETICS",
    "auth_version": "2.0_password"
  }

//Kubernetes provider

  {
    "id": "ad7c1a70-105c-4827-8d6e-f3e8fca2e74c",
    "name": "GabrieleOnK8S-ENG_k8s",
    "class": "benchsuite.stdlib.provider.kubernetes.KubernetesProvider",
    "driver": "kubernetes",
    "description": "Autogenerated by the benchsuite-pledger synchronizer",
    "grants": {
      "viewer": "org:public",
      "owner": "usr:gabriele"
    },
    "metadata": {
      "pledger-service-provider": "5",
      "pledger-infrastructure": "1",
```

```
      "pledger-project": "6"
    },
    "user_token": "********",
    "namespace": "core",
    "api_server_url": "https://192.168.111.52:6443"
  }
```


3.4.2 Workload

The workload object models the test to execute on the target infrastructure. It contains all the information to execute the test program. There are two different types of workload objects: Bash workloads and Docker workloads. In Bash workloads, the test is defined through a series of bash scripts to be executed to install, execute and remove the test. In Docker workloads, instead, the test is defined by a Docker image that contain the test and will run it when execute. The Docker format is simpler to use and to define (no compatibility issues with multiple platform/Operating System (OS) scripts) and also is the only one supported by Kubernetes providers.

The code below shows a Docker workload as returned by the APIs.


```
{
    "workload_parameters": {
      "OPTIONS": "--cpu-max-prime=20000 --events=10000",
      "TEST": "cpu",
      "N_THREADS": "1 2 4 8 16 32",
      "args": "-c\nfor i in $N_THREADS; do echo \"###$TEST###$i###\"; sysbench
$TEST run $OPTIONS --time=0 --threads=$i; echo \"###END###\"; done",
      "cmd": "/bin/sh",
      "image": "severalnines/sysbench"
    },
    "categories": [
      "cpu"
    ],
    "grants": {
      "executor": "org:public"
    },
    "workload_name": "docker_cpu_10000",
    "tool_name": "SysBench",
    "id": "90f8ea37-4881-4edf-89b3-0bb97bf9993e",
    "description": "first docker workload",
    "class": "benchsuite.stdlib.benchmark.docker_benchmark.DockerBenchmark",
    "parser": "benchsuite.stdlib.benchmark.parsers.SysbenchResultParser"
  }
```

3.4.3 Schedule

The schedule object represents the willing of executing a given test (or multiple tests) on a given provider with a given frequency of time. The schedules are used by the Scheduler component to trigger the execution of tests. The code below shows a schedule object as returned by the APIs. In addition to the provider, workloads and the time interval, there is also a flag to specify whether the schedule is active or not and the visibility of the produced results (i.e., "reports-scope").

```
{
  "id": "c1c42788-974a-4817-961d-6f4ccced236a",
  "name": "Testing-GabrieleOnK8S-ENG_k8s",
  "interval": {
    "minutes": "45"
  },
  "tests": [
    "90f8ea37-4881-4edf-89b3-0bb97bf9993e"
  ],
  "active": true,
  "grants": {
    "executor": "usr:gabriele",
    "owner": "usr:gabriele"
  },
  "provider_id": "ad7c1a70-105c-4827-8d6e-f3e8fca2e74c",
  "username": "gabriele",
  "reports-scope": "usr:gabriele"
}
```

3.4.4 Execution

The execution object represents the execution of a schedule. It contains a reference to the schedule that originated the execution and the status of single tests in the execution as well as overall status of the execution. The code below shows an execution object as returned by the APIs.

```
{
  "status": "OK",
  "runs_status": {
    "ok": [
      "608c91a31c765d5aac06b382"
    ],
    "error": []
  },
  "starttime": "2021-04-30 23:22:52.686000",
  "endtime": "2021-04-30 23:24:21.067000",
  "id": "13679053-28af-48ea-9b57-3295eb6fb6e9",
  "schedule": {
    "id": "c1c42788-974a-4817-961d-6f4ccced236a",
    "name": "Testing-GabrieleOnK8S-ENG_k8s"
  }
}
```

4 Installation and usage guides

4.1 Requirements

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

- Kubernetes cluster v1.15+ [3] with two persistent volumes for MongoDB and InfluxDB databases;
- Helm v3+ [12];
- (Optional) a Keycloak [4] 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 [8] instance that will be used to send and receive model updates and send metric reports.

4.2 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.res.eng.it/benchsuite/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.res.eng.it/benchsuite/benchsuite-helm.

4.3 Usage

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/. A short tour on the application functionalities is included in this document in section 5.3.

4.4 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.

4.5 Source code repository

The Benchmarking Suite source code is split in multiple Git repositories that are publicly available at https://gitlab.res.eng.it/benchsuite.

5 Demonstration

5.1 Scenario description

Two scenarios have been prepared for the demonstration of the Benchmarking Suite: i) a "stand-alone" scenario where the Benchmarking Suite is used alone and ii) an "integrated" scenario where the integration of the Benchmarking Suite with the other Pledger's components is demonstrated.

The basic functionality of the Benchmarking Suite can be demonstrated stand-alone in the following way (see screenshots for each test in section 5.1):

1. The user loads the web-based Benchmarking GUI application;
2. The user logs-in (this process involves the redirect to Keycloak for the authentication);
3. The user creates a new *Provider*: in case of Kubernetes, she will be asked to insert the API Server endpoint, user token and namespace;
4. The user creates a new *Schedule*: 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;
6. The user can verify that new executions of the same test will be scheduled accordingly with the repetition frequency set in the Schedule configuration:
7. 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.

In order to demonstrate the integration with the Pledger's Configuration Service which integrates and automates the benchmarking process in the Pledger platform, the following steps can be followed:

1. The user creates a new *Project* in the Configuration service. This will trigger a synchronisation of the Benchmarking Suite model that will create a *Provider* with the same data of the Project created by the user and a default Schedule. This step replaces steps 3 and 4 in the stand-alone scenario;
2. The user can check the execution of tests and analyse the results in the Benchmarking GUI following steps 5, 6 and 7 in the stand-alone scenario;
3. When new results are available, they will be synchronised with the Configuration service and will appear in the Benchmarking Report page.

5.2 Validation and verification

This first iteration of Benchmarking subsystem addresses the MVP requirements for this first iteration of the tool. The System User Cases (SUC) that are enabled by the release of this version of the tool are the following:

- SUC.18 IaaS self-assessment;
- SUC.21 get IaaS evaluation;
- SUC.22 compare different IaaS providers;
- SUC.25 select and run benchmarking test(s) from library;

These SUC were defined in D2.3 PLEDGER Overall Architecture [1].

Validation and Verification of the Benchmarking Suite will be carried out during the project by executing the supported benchmark tests for each of the project's testbed infrastructure (Cloud and Edge). Accuracy of results of single tests will be evaluated in the project when enough results for each test and infrastructure will be collected.

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 Key Performance Indicators (KPI):

- **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;
- **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.

5.3 Demo

This section collects few screenshots of the Benchmarking Suite GUI that illustrate the demonstration steps defined in section 5.1.

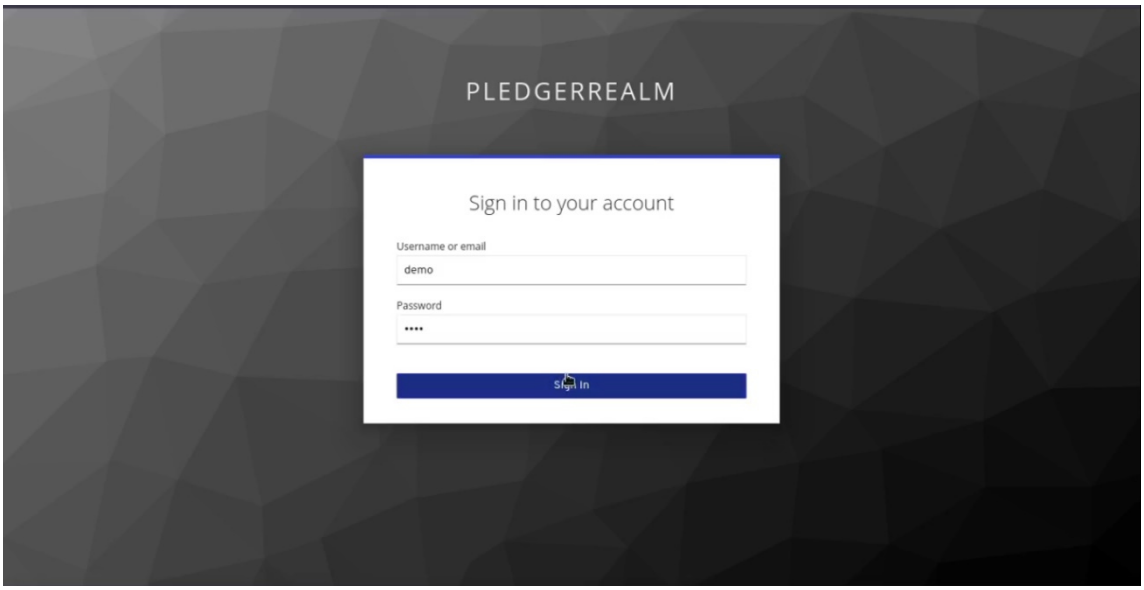

Figure 6: Benchmarking Suite log-in page

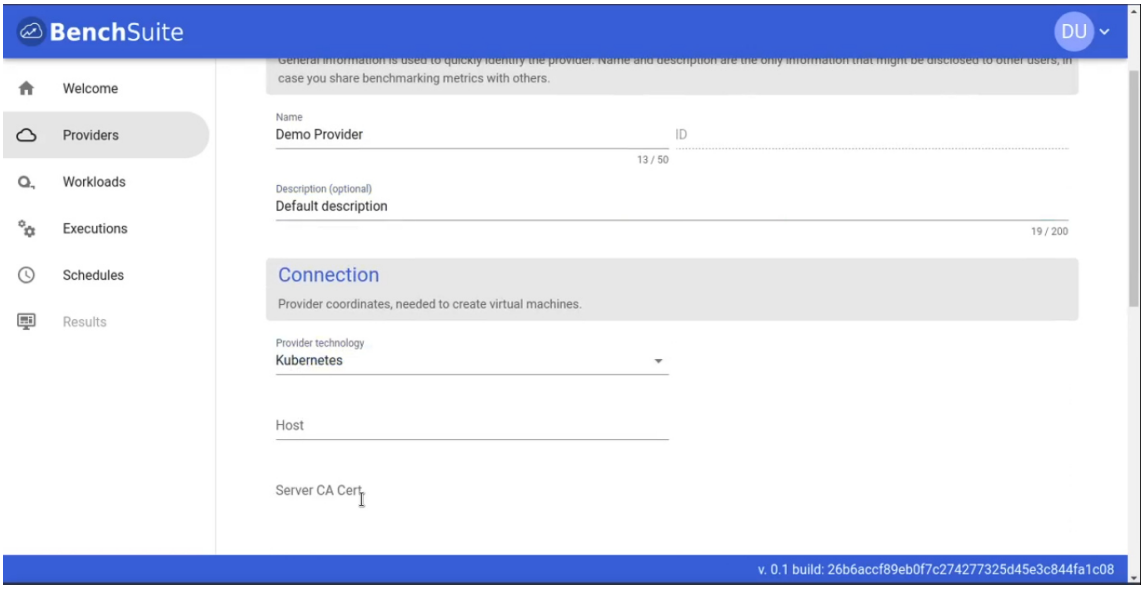

Figure 7: Benchmarking Suite Provider creation page

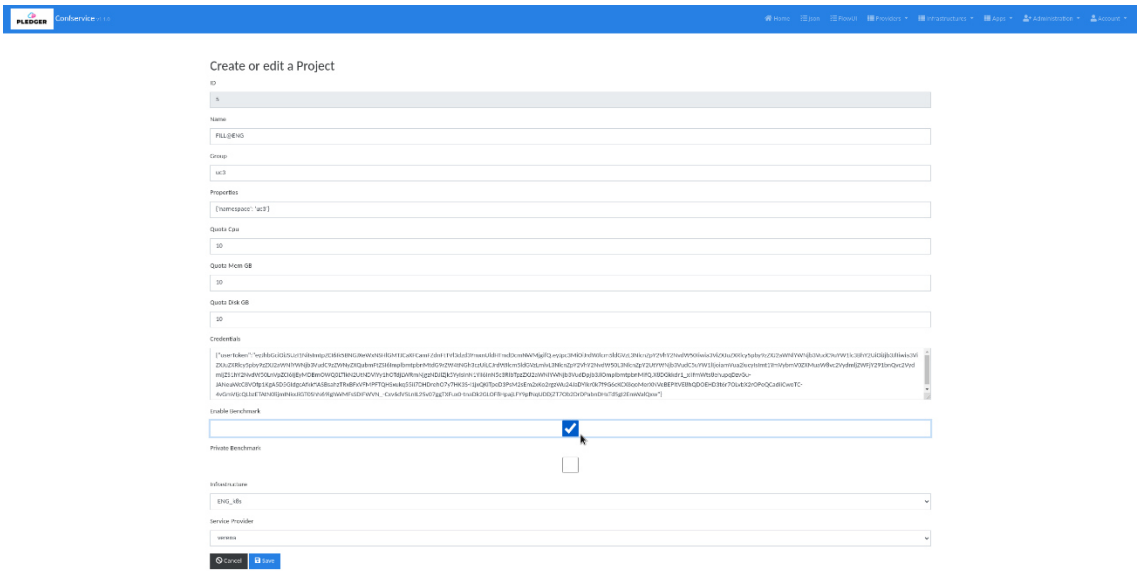

Figure 8: Configuration Service Project creation

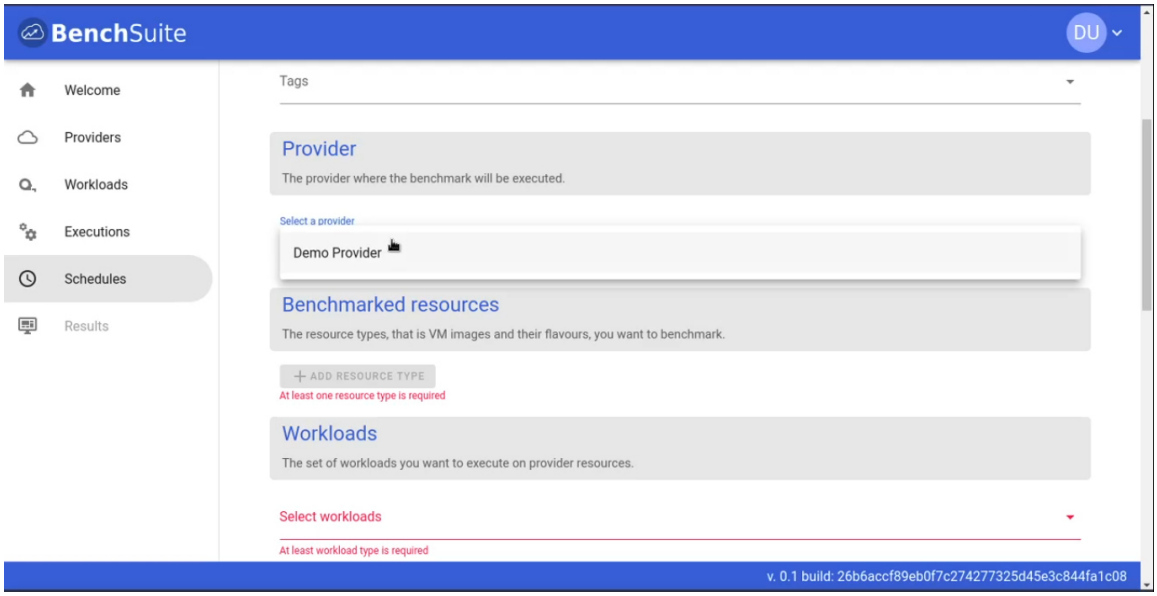

Figure 9: Benchmarking Suite Schedule creation page

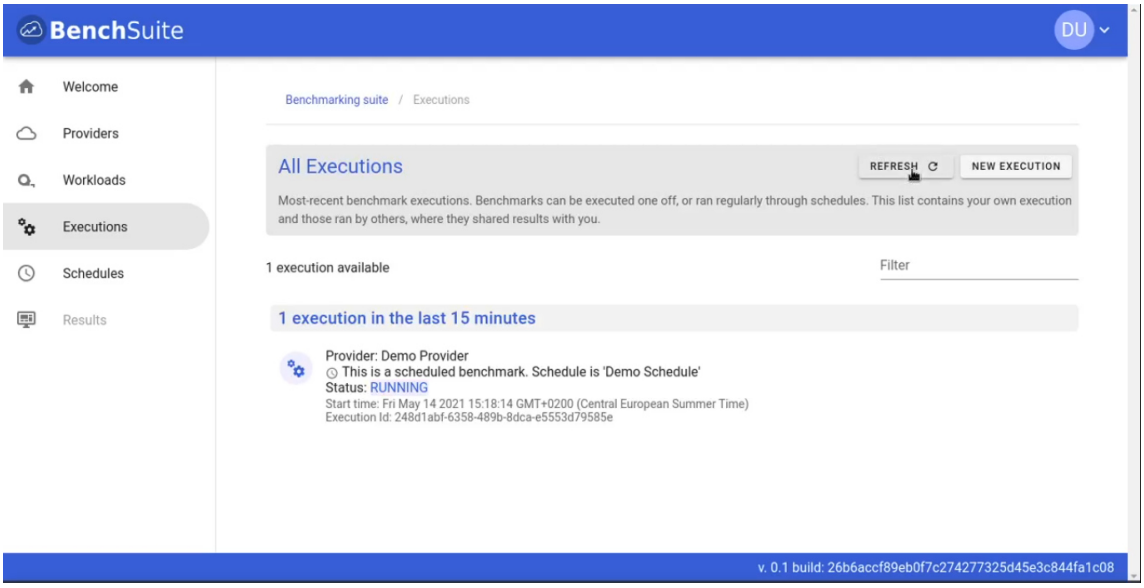

Figure 10: Benchmarking Suite Executions page

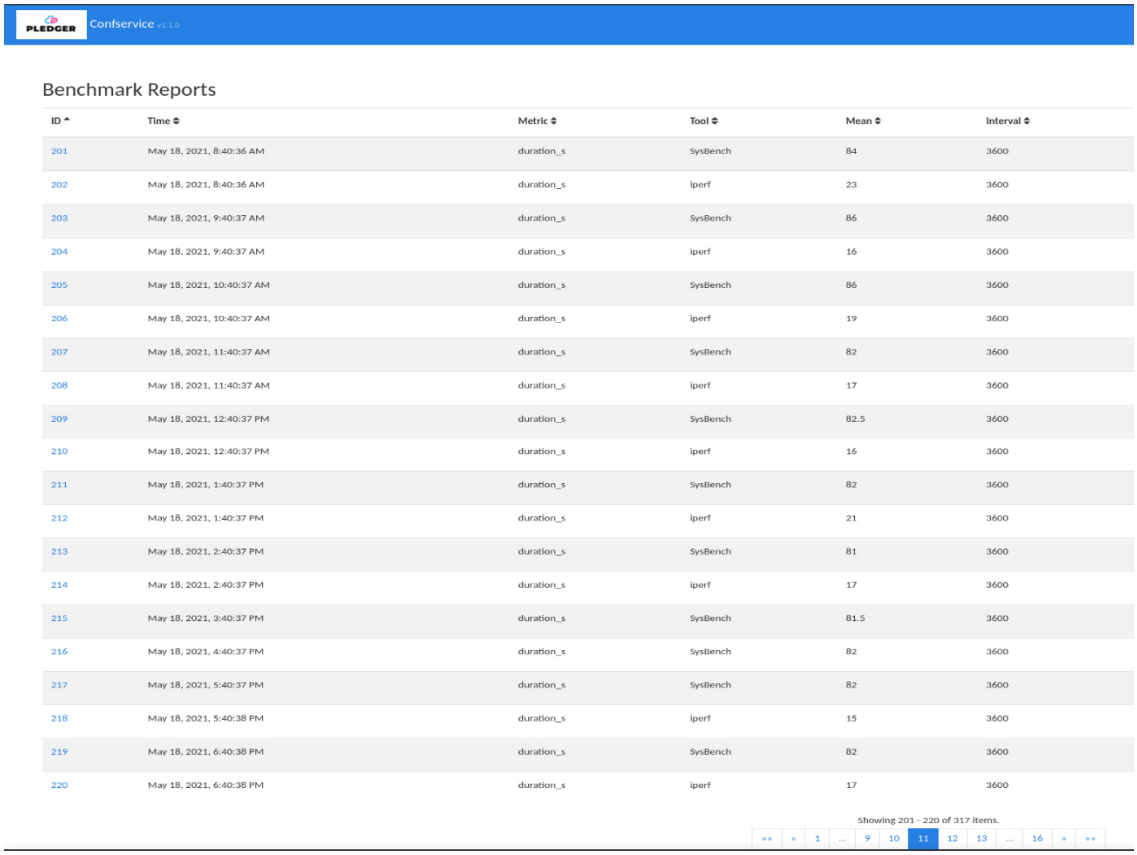

Figure 11: Benchmarking reports sent to the Configuration service

Figure 12: Benchmarking Suite Results page 1

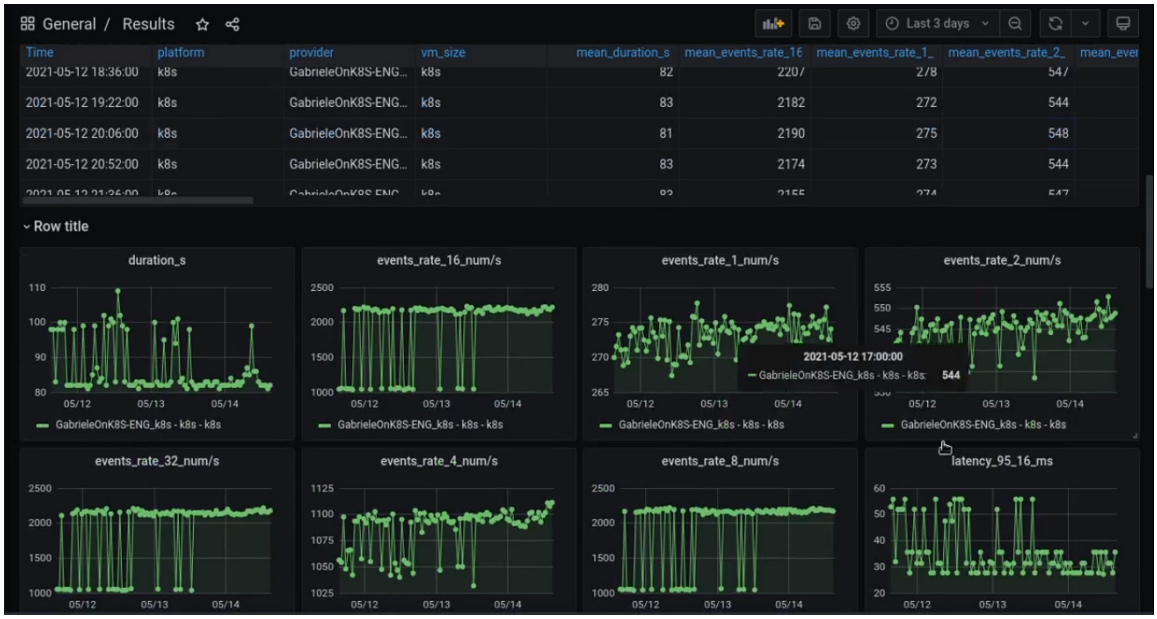

Figure 13: Benchmarking Suite Results page 2

6 Conclusions

This document accompanies the release of the components in the Benchmarking subsystem developed in the context of project's Task "T3.1 - Performance Measurements and Classification". The document presented the main functional and technical aspects of the prototype as well as guidelines to use it. The prototype released is based on the "Benchmarking Suite" asset adapted and evolved by Engineering Ingegneria Informatica S.p.A. (ENG).

The development of the tool followed the requirements and the architecture identified in deliverable D2.2 [13] and D2.3 [1]. The main extensions include the implementation of authentication and multitenancy, time-series backend, support for Kubernetes clusters and Docker-packaging of benchmarks. The tool has been integrated with the Configuration service (WP4) to retrieve infrastructures configuration and to send benchmarking reports. The current version of the tool enables following System Use Cases: IaaS self-assessment (SUC.18), get IaaS evaluation (SUC.21), compare different IaaS providers (SUC.22), select and run benchmarking test(s) from library (SUC.25).

The Benchmarking Suite has been released as an open-source software (Apache 2.0 License) available at https://gitlab.res.eng.it/benchsuite with an administration and user guide available online at https://benchmarking-suite.readthedocs.io/. The Benchmarking Suite can be easily deployed and configured in any Kubernetes cluster using a specific Helm chart developed during the project.

The current version of the prototype will be deployed in the project's testbeds and validated in the context of WP5. The development of the remaining requirements (in particular, the user provided benchmarks and customizable dashboards) will be completed during the second development iteration and will be documented in deliverable D3.4 at project month 30 (M30).

7 References

[1] PLEDGER. D2.3 - Pledger Overall Architecture. Voutyras, Orfefs. 2020

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

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

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

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

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

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

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

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

[10] GOLANG home page. https://golang.org/, retrieved 2021-05-27.

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

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

[13] PLEDGER. D2.2 - Pledger Requirements Analysis, Iadanza, Francesco. 2020