CDSE TESTING AND PLUGIN

Author: **Virginia Valeri**

Abstract

These Technical Report provides a comprehensive overview of the CDAB Simple Plugin, a lightweight, web-based interface designed for benchmarking data providers using the Copernicus Sentinels Data Access Worldwide Benchmark Test Suite. The plugin facilitates rapid and efficient execution of benchmark tests, focusing on simplicity and user accessibility. This report details the system requirements, installation processes, and step-by-step usage instructions, ensuring ease of use for diverse operating systems and user backgrounds. Significant emphasis is placed on the results visualization mechanisms employed by the plugin, which include automated Google Sheets for data recording and graphical representations Additionally, an interactive map visually represents the global distribution of test usage, enhancing the understanding of the plugin's worldwide application and performance. Through detailed analysis and discussion, the report highlights the plugin's effectiveness in optimizing benchmark testing procedures and exemplifies a successful integration of web technologies with benchmarking tools, aiming to streamline and enhance the user experience in data accessibility testing.

Key-words: CDAB Simple Plugin, benchmark testing, web-based interface, data visualization , system requirements, installation processes, user accessibility, web technologies

Contents

1 Introduction

The Copernicus Sentinels Data Access Worldwide Benchmarking service aims at providing impartial and reliable information on the Sentinels data accessibility, providing a comprehensive and factual overview of the conditions experienced by users in Europe and beyond. The Sentinel Data Access Worldwide Benchmark project, operating under an assignment from the European Space Agency and funded by the European Union in the frame of the Copernicus Programme, underscores our commitment to enhance data accessibility across various platforms. This report stems from the broader initiative of the Copernicus Sentinels Data Access Worldwide Benchmark, documented in the Service Design Document (poi sotto metto il link del SDD).

Originally, the Copernicus Sentinels Data Access Worldwide Benchmark Test Suite served as a foundational tool for evaluating the performance of various Copernicus Data Provider targets, including CREODIAS, Mundi Web Services, ONDA, and several others. Developed collaboratively by Exprivia and TerraDue, this suite was designed to encompass a broad spectrum of test scenarios, addressing multiple functionalities across these diverse target sites. While accessible through a public Docker image and enabling complex benchmarking operations, the original process often required extensive setup and execution time.

The CDAB Test Suite is an ensemble of Software components to measure metrics in a set of Test Cases organised in Test Scenarios. It is provided in a public repository available on GitHub at this URL:

- https://github.com/esa-cdab/cdab-testsuite

Although accessible through a public Docker image enabling complex benchmarking operations, the initial setup and execution processes posed significant challenges:

- **Technical Requirements**: Users were required to have Python installed with all necessary dependencies and packages. The need to manage these installations along with other software meant that users needed substantial technical skills to utilize the suite effectively.
- **Operational Complexity**: Operating the test suite involved running multiple lines of command in the terminal, which could lead to errors in typing and command execution, complicating the user experience.
- **Data Handling**: The results from the test scenarios were downloaded as JSON files—a format best suited for those with technological expertise, thus limiting accessibility to users with less technical backgrounds.

The suite supported various Data Access Hubs and DIAS platforms, including the Copernicus Data Space Ecosystem and others, structured into both local and remote test scenarios managed via command-line tools provided in the public repository.

Recognizing the need for a more streamlined and user-friendly approach, the CDAB Simple plugin was created. The Plugin, built on the robust original work of the Test Suite, is designed to streamline the process of data benchmarking through an intuitive and efficient web interface, facilitating immediate and straightforward interactions for users worldwide. By integrating functionalities that allow users to execute predefined test scenarios with ease, this plugin addresses the pressing need for accessible and reliable data benchmarking tools.

This initiative is part of a broader effort to provide reliable, timely, and accessible data services to users globally, ensuring that the benefits of the Copernicus program are maximized. Through this report, we aim to detail the functionalities, benefits, and user interactions provided by the CDAB Simple Plugin, reinforcing our commitment to improving the benchmarking process in the data space ecosystem.

1.1 Overview of CDAB Simple Plugin

1.2 Objectives and Scope

The primary objective of the CDAB Simple Plugin is to facilitate efficient and effective benchmark testing of data providers, focusing on simplicity and user accessibility.

This tool aims to minimize the complexity involved in executing test scenarios, making it accessible to users with varying levels of technical expertise. The scope of this plugin includes:

- Streamlining the benchmark testing process for Copernicus data providers.
- Providing a comprehensive, easy-to-navigate interface.
- Ensuring compatibility across diverse operating systems.
- Enhancing data visualization through automated tools and interactive maps.
- Delivering reliable and scalable performance for global users.

1.3 Report Structure

This report is structured to provide a thorough overview of the CDAB Simple Plugin, from system requirements and installation processes to detailed usage instructions and results visualization. The subsequent sections are organized as follows:

Creating the Plugin: Explaining the Development Process

System Requirements: Details the technical prerequisites for installing and running the plugin.

Installation Process: Guides through the step-by-step procedure to set up the plugin effectively.

Using the Plugin: Explains the functionalities and operational guidelines for users.

Results Visualization: Describes the mechanisms for visualizing test results, including graphical representations and the use of interactive maps.

Discussion: Analyzes the performance and effectiveness of the plugin, emphasizing its impact on optimizing benchmark tests.

Conclusion: Summarizes the key findings and potential areas for further enhancement.

2 Background

2.1 Importance of Copernicus Sentinels Data Access Worldwide Benchmark Test Suite

The importance of the Copernicus Sentinel Data Access Worldwide Benchmark Test Suite is outlined comprehensively in the provided Service Design Document. This benchmarking suite plays a critical role in ensuring the accessibility, reliability, and performance of the data provided by the Copernicus Sentinel satellites across various platforms and services.

The benchmark test suite provides impartial and reliable information on the accessibility of Sentinel data, ensuring that users globally can depend on consistent and predictable data services. It allows for a comprehensive evaluation of the conditions experienced by users, including data availability, quality, and speed, which are critical for applications that rely on timeliness and accuracy.

By systematically testing and reporting on various data access points, the benchmark suite helps identify and address potential issues, continuously enhancing the user experience and supporting the development of a user-friendly interface for easier and faster data access. The benchmark results are presented to the Copernicus governance bodies, providing essential insights into the performance of data distribution and access services, supporting informed decision-making regarding infrastructure improvements and policy development.

The benchmark test suite is crucial for the ongoing evolution and adaptation of the Sentinel data access services, helping assess the effectiveness of current services and plan future upgrades or expansions to meet changing user demands. Regular benchmarking and reporting foster greater engagement with stakeholders, including data providers and users, aligning services with user needs and enhancing the overall utility of the Copernicus program.

The suite extends its benchmarking activities globally, ensuring that the Sentinel data services meet international standards and serve a global user base effectively.

2.2 Overview of the Web-based Interface

The web-based interface is a comprehensive, user-friendly platform that excels in real-time interactions, automated test execution, results visualization, accessibility, security, reliability, and scalability.

The interface is designed with **user accessibility** in mind. Its clean and intuitive layout ensures that users can navigate the platform easily, regardless of their technical experience. It streamlines complex processes, enabling users to initiate and monitor benchmarks tests with minimal input and effort.

A standout feature of this interface is the ability to support **real-time interactions**, it ensures that the interface can effectively handle many dynamic data inputs and provide users with up-to-date information. Users can trigger test scenarios, monitor ongoing test statues and receive immediate feedback on test execution.

Furthermore, the interface allows for **automated test execution**, which significantly boosts the efficiency of the benchmarking process. Users can schedule tests, select specific test parameters, and automate repetitive tasks without manual intervention. This automation not only speeds up the testing process but also ensures consistency and accuracy in test execution.

Another significant feature of the web-based interface is its robust **results visualization tools**. It integrates functionalities such as automated Google Sheets for data recording and dynamic graphical representations to visualize test results effectively. Users can interpret test results through charts, graphs, and tables that are updated in real time. Additionally, an interactive map displays the global distribution of test usage, providing a geographical context to the data, which enhances understanding of the plugin's application and performance worldwide.

The interface is developed to be **accessible** on various devices and compatible with multiple operating systems, ensuring that users can access the platform from anywhere and at any time. This accommodates a diverse user base with different hardware and software environments.

Finally, the web-based interface is built to scale. As the number of users and the volume of data grow, the interface can handle increased loads without compromising performance. This scalability is essential for future expansions and for accommodating larger datasets as the Copernicus program evolves.

3 System Requirements

Ensuring that the CDAB Simple Plugin runs efficiently and effectively involves meeting specific system requirements. This section provides details on the supported operating systems and software dependencies necessary to install and operate the plugin optimally.

3.1 Supported Operating Systems

The CDAB Simple Plugin is designed to be highly compatible and performant across various computing environments. The following operating systems are officially supported:

Windows: The plugin is compatible with Windows 10 and newer versions. It has been rigorously tested in Windows environments to guarantee smooth operation, particularly for users with administrative access required to install necessary components.

macOS: For Mac users, the plugin supports macOS and newer versions.

Linux: The plugin also caters to Linux users, supporting distributions such as Ubuntu and other popular variants. This broad Linux compatibility is crucial, as Linux is often preferred in academic and server environments for its robustness in handling data-intensive processing tasks.

3.2 Software Dependencies

To ensure that the CDAB Simple Plugin runs seamlessly, only a software dependency must be installed:

Docker Desktop: Download and install Docker from https://www.docker.com/get-started/

The executable created includes all necessary dependencies and packages, eliminating the need for users to install them separately. The developer has bundled everything required in the backend, ensuring the application runs smoothly without additional setup.

4 Creating the Plugin

4.1 Overview of the Project Strcuture

Overview of the Project Structure

- **Results**: Directory for storing output results from test executions.
- **Static**: Static files like CSS for styling, JavaScript for client-side logic
    - **CSS**: Style sheets for the web application.
    - **JS**: JavaScript files to add interactivity to web pages.
- **Templates**: Contains HTML files that serve as the frontend for the Flask application.
- **app.py**: The main Python script that initializes the Flask app and defines routes and their functionalities.
- **config.yaml**: Configuration file used to manage settings or parameters for the application.

Key Features and Functionalities

Docker Integration

- The application integrates Docker, allowing tests to run in isolated environments. This ensures that tests do not interfere with each other and can be set up with specific configurations without affecting the host system.

Test Execution and Management

- Tests are managed and executed through the Flask application, which interfaces with Docker to control the lifecycle of containers. The application handles test configuration, execution, and cleanup.

Results Handling

- Test results are captured and possibly stored temporarily in JSON format before being processed or displayed. This modular approach allows for flexibility in how results are handled, displayed, or stored.

Dynamic Web Interface

- The application uses Flask templates to serve dynamic web content. Users can interact with the system through forms to configure and initiate tests, view results, and interact with data visually through embedded plots and maps.

5. Google Sheets Integration

- The application interacts with Google Sheets using the Google Sheets API to store and retrieve test results. This allows for persistent storage of test data and easy sharing and analysis.

Code Explanation and Functions

- **Flask Routes and Views**: In app.py, various routes handle different parts of the application workflow, from starting tests (/run_tests) to displaying results (/results) and error handling (/error).
- **Error Handling**: The application includes error handling to manage and respond to issues during test execution, particularly around Docker operations.
- **Data Visualization**: Results visualization is managed through templates that incorporate data plotting directly into the web pages, enhancing user interaction and engagement.

HTML and JavaScript Integration

- **HTML Templates**: The application uses HTML templates to present interactive forms (metrics_form.html), results (results.html), and historical data (history.html). These templates are populated dynamically using data passed from Flask.
- **JavaScript**: The application leverages JavaScript for enhancing the interactivity of the web pages, such as handling form submissions, updating content dynamically, and integrating third-party libraries like Leaflet for mapping functionalities.

4.2 .PY EXPLANATION

app.py - Flask Application Setup and Route Definitions

The app.py file is the backbone of your Flask application, managing the web server and routing user requests to appropriate handlers. It defines the structure of the web application and how it responds to various client actions.

5 Key Components and Functions:

1. **Initialization and Configuration**:

- **Flask instance**: An instance of the Flask class is created, initializing the application.
- **Secret key setup**: Configures a secret key for session management, crucial for maintaining secure user sessions and CSRF protection.

2. **Route Definitions**:

- **Home (/)**: Serves the main page of the application (index.html), which is the starting point for users.
- **Run Tests (/run_tests)**: Handles the form submission from the index.html page, where users configure and start new tests. This route retrieves form data, validates it, and calls the run_tests function to execute the tests.
- **Results (/results)**: Displays the outcomes of tests after processing. It pulls data from the session and uses it to populate results.html.
- **Metrics Form (/metrics_form)**: Manages both GET and POST requests for entering and submitting detailed test metrics, which are then sent to Google Sheets.
- **History (/history)**: Fetches historical test results from a Google Sheet and displays them on history.html, allowing users to view past data.

3. **Error Handling**:

- **Error routes (/error)**: Provides user-friendly error messages via error.html for different types of application failures, improving the user experience during issues.

6 Important Functions:

- **run_tests_route()**: Validates user input and coordinates the execution of tests. This function demonstrates how backend validations are handled and how external function calls are integrated into Flask routes.
- **write_to_sheet()**: This function interacts with the Google Sheets API to write test metrics to a spreadsheet, showcasing the application's capability to integrate with external services for data storage.

functions.py - Utility Functions Supporting Test Execution

The functions.py file contains utility functions that support the main Flask application, particularly focusing on managing Docker environments and handling test execution.

Key Functions:

1. **run_tests()**:
   - Initiates Docker containers, executes specified tests using Docker commands, and handles the collection of results from Docker containers.
   - Demonstrates complex integration with Docker, showing how external commands can be run and managed from within a Python application.
2. **generate_results()**:
   - Processes JSON files containing test results, extracts relevant data, and formats it for display in the Flask application.
   - Illustrates data processing and the transformation of raw test results into user-friendly formats.
3. **Docker Management**:
   - **is_docker_running()**: Checks if Docker is running on the host system, ensuring that the application can proceed with test executions that require Docker.
   - **check_container_exists()**: Verifies the existence of a Docker container before attempting operations, preventing errors related to container management.
4. **Visualization**:
   - **plot_time_series() and plot_scatter()**: Generate visual representations of test results, such as time series and scatter plots. These functions use Matplotlib to create plots, highlighting the application's capability to provide graphical data insights to user

6.1 HTML PAGES EXPLANATION

INDEX.HMTL- Home and Test Configuration Page

This is the main landing page of the application. It serves as the entry point where users can initiate new tests and view historical results. It contains:

- **Header Section**: Displays the application title and introductory text, guiding users on how to proceed.
- **Test Configuration Form**: Users can enter the name of a Docker container and select from predefined test scenarios and service providers. The form includes checkboxes for selecting specific tests (TS01, TS02) and service providers (cdse).

- **Interactive Elements**: Tooltips provide additional information about each test scenario, enhancing user understanding without cluttering the interface.
- **Navigation Buttons**: Links to run tests directly or navigate to the history page to view past results.
- **JavaScript Enhancements**: Includes scripts for smooth scrolling and other interactive behaviors to enhance user experience.

RESULTS.HTML- Results Display Page

After tests are executed, this page displays the outcomes. It is crucial for users to understand the performance metrics from their test scenario

- **Results Table**: Dynamically populated with data from test executions, showing metrics like response times, error rates, and concurrency levels.
- **Navigation Options**: Provides a link back to the home page or to publish detailed results to Google Sheets, fostering an interactive flow within the application.

METRCIS_FORM.HMTL - Metrics Submission Form

This page is used for entering detailed metrics data that users wish to store in Google Sheets:

- **Form Fields**: Allows users to input data such as location, company, date, and specific metrics related to the tests they ran. Users have only to add the location, the date and the company they work for, the metrics are already precompiled for the sake of simplicity.
- **Data Validation**: Ensures that all inputs are correctly formatted and required fields are filled, ensuring data integrity.
- **Submission Buttons**: Facilitates the submission of data to Google Sheets, with user feedback indicating successful data transfer.

HISTORY.HTML - Historical Data Viewing Page

A page dedicated to viewing historical results, enabling users to analyze past performance data over time:

- **Data Table**: Lists all past results with details on test parameters and outcomes, allowing users to track changes and trends over time.
- **Interactive Map**: Integrates with Leaflet to display geographical data points related to test executions, enhancing the contextual understanding of data.
- **Graphical Plots**: Time-series plots or scatter plots generated from historical data, providing a visual summary of performance metrics.

- **Back Navigation**: Offers a button to return to the home page, maintaining easy navigation throughout the application.

These HTML pages constitute the core of the user interface for the CDAB Simple Plugin, each tailored to handle different aspects of the user's journey from configuring tests, viewing results, entering detailed metrics, to analyzing historical data.

7 Installation and Setup

Installing and setting up the CDAB Plugin involves several steps to ensure the application is configured correctly for optimal performance. This section outlines the process for downloading the executable, setting up the environment, and running the plugin.

7.1 Downloading and Running the executable

To begin using the CDAB Simple Plugin, follow these steps to download and run the executable:

1. **Download the Executable:**
   - Navigate to the official repository of the CDAB Simple Plugin on GitHub https://github.com/Virginia555/cdseesa-Plugin
   - Click on the download link to obtain the .zip file containing the executable and all the related files.
2. **Extract the Files:**
   - Once the download is complete, extract the .zip file to a desired location on your computer system. This will create a directory containing the executable and all necessary dependency files.

3. **Run the Executable:**

- On Windows, double-click on app.exe, found in the *Dist* folder to launch the application.
- On macOS and Linux, open a terminal, navigate to the directory containing the executable, and run ./app to start the server.
- Ensure that any firewall or security software is configured to allow the application to run and access the network if needed.
- Running the executable will start the web server hosting the CDAB Simple Plugin's interface, typically accessible via a web browser at ***http://127.0.0.1:5000.***

7.2 Environment Setup

If you decide to run the executable, you do not have to continue reading this paragraph, because the dependencies are already installed.

This setup includes installing necessary software dependencies, configuring system settings, and ensuring that all required services are running. Here's how to set up the environment:

1. **Install Required Software**:
   - **Python**: Ensure that Python is installed on your system, as it is required to run the Flask application that serves the CDAB Simple Plugin.
   - **Docker**: Install Docker if it's not already installed. Docker is necessary for containerization, which isolates the application and its dependencies from the system.
2. **Alternative Setup using an IDE**:
   - Instead of running the executable, you can also download the .zip folder of the plugin, extract it, and open it in an IDE like Visual Studio Code.
   - Ensure that you have Visual Studio Code or another IDE installed that can handle Python projects.
3. **Configure System Settings**:
   - Adjust any firewall settings to allow traffic on the necessary ports, typically port 5000 for local development.
   - Check your system's security settings to ensure they don't block the execution of scripts from the plugin's directory.
4. **Install Python Dependencies**:

- Navigate to the project's root directory in your terminal or command prompt.
- Run *pip install -r requirements.txt* to install all required Python packages listed in the requirements.txt file.

7.3 Running the Plugin

Once the environment is properly set up, you have two options to run the CDAB Simple Plugin:

Option 1: Using the Executable

1. **Run the Executable**:
   - On Windows, double-click on app.exe found in the Dist folder to launch the application.
   - A command prompt window will open, and you should see a message like Running on http://127.0.0.1:5000.
   - Click on this link, or enter it into your web browser to access the web interface of the CDAB Simple Plugin.

Option 2: Using an IDE or Command Line

1. **Start the Application from Source**:
   - Open a terminal or command prompt and navigate to the directory where you have the plugin's files (after extracting the .zip file).
   - Type python app.py to start the Flask server.
   - Similar to running the executable, you should see Running on http://127.0.0.1:5000 in your terminal.
   - Click the link or manually enter it into your web browser to open the web interface.

8 Using the Plugin

8.1 Overview of Web Interface Functionality

8.2 Selecting and Executing Test Scenarios

Executing tests is a straightforward process that involves several user-friendly steps:

Accessing the Test Selection Page:

- Users start by navigating to the 'Test Scenarios' section of the web interface, accessible from the main dashboard.
- **User Interface Features**: The interface is designed to be intuitive, displaying all available test scenarios such as TS01 and TS02. Each scenario includes a brief description to help users choose based on their specific testing needs.

Configuring Test Parameters:

- For selected scenarios, users can set parameters such as the target data provider, scale of testing, and specific data types or time ranges.
- **Customize Settings**: Optional settings allow for customization of tests, such as adding multiple providers or adjusting the depth of data retrieval.

Executing the Test:

- **Initiate the Test**: Once the scenarios and parameters are set, click the 'Run Test' button to start the testing process.
- **Real-Time Monitoring**: The system processes the request and begins the test. Users can monitor the progress through a real-time status bar or detailed log output provided in the interface.

Viewing Preliminary Results:

- Once a test completes, initial results are displayed in a summary format directly within the interface.
- Users can view detailed logs, download result files, or visualize data through integrated charts and graphs directly from the test completion screen.

9 Test Scenarios and Execution

9.1 Description of Test Scenarios TS01 and TS02

The Test Suite is orgnaized around a series of Test Scenarios chaining a set of test Cases. The **Test Scenarios** are designed as a sequence of basic Test Cases covering one or several functionalities of the Target Site in order to reproduce a typical user operation on the target site.

10 TS01: Simple Data Search and Single Download

- **Objective:** Evaluate the performance and reliability of retrieving a single dataset or file from the target data provider.
- **Activities Involved:**
    - Perform a search query for a specific dataset.
    - Initiate a download of the selected dataset.
    - Measure the time taken from query to download completion and log any errors encountered.
- **Metrics Measured:**
    - Response time
    - Data integrity
    - Success rate
- **Test Scenario Details:**
    - **Applicable Interface Type:** Application programming interface (API)
    - **Indicative Test Frequency:** Hourly
    - **Load Configuration:** Load factor = 3, max parallelism = 1-10 (varies by target site limitations)
    - **Orchestration:** Test cases sequence - #TC101 → #TC201 → #TC301
    - **TC101: Service Reachability**

      This test performs multiple concurrent remote HTTP web requests to the front endpoint of the target site. It measures, among other metrics, the average and peak response times.
    - **TC201: Basic Query**

This test performs a simple filtered search (e.g. by mission or product type) and verfies whether the results match the specified search criteria. The test client sends multiple concurrent remote HTTP web requests to the front OpenSearch API of the target site using the OpenSearch mechanism to query and retrieve the search results. Searches are limited to simple filters (no spatial nor time filters) established randomly on the missions dictionary.

Among the obtained metrics are the average and peak response times, the number of results and the size of the responses.

- **TC301: Single Remote Online Download**

  This test evaluates the download service of the target site for online data. The test client makes a single remote download request to retrieve a product file via a product URL.

  Among the metrics the test obtains is the throughput of the downloaded data.

- **Relevant Metrics:**
  - Average response time (M001)
  - Peak response time (M002)
  - Error rate (M003)
  - And others related to data size and throughput

TS02: Complex Data Search and Bulk Download

- **Objective:** Assess the system's capacity to handle complex queries and manage multiple simultaneous downloads effectively.
- **Activities Involved:**
  - Conduct searches using advanced query parameters that encompass multiple data types and temporal ranges.
  - Simultaneously download multiple datasets.
  - Evaluate the performance of the data provider's infrastructure under increased load.
- **Metrics Measured:**
  - Throughput

    - Error rate
    - Average and peak response times
- **Test Scenario Details:**
    - **Applicable Interface Type:** Application programming interface (API)
    - **Indicative Test Frequency:** Daily
    - **Load Configuration:** Load factor = 3, max parallelism = 1-10 (varies by target site limitations)
    - **Orchestration:** Test cases sequence - #TC101 → #TC202 → #TC302
    - **TC101: Service Reachability**

      This test performs multiple concurrent remote HTTP web requests to the front endpoint of the target site. It measures, among other metrics, the average and peak response times.
    - **TC202: Complex Query (Geo-Time Filter)**

      This test performs a more complex filtered search (e.g. by geometry, acquisition period or ingestion date) and verifies whether the results match the specified search criteria. The test client sends multiple concurrent remote HTTP web requests are sent to the front catalogue search API (preferably OpenSearch API) of the target site using the search mechanism to query and retrieve the search results. N queries are prepared with all filters (spatial and time filters included) and composed with random filters from the missions dictionary.

      The obtained metrics are the same as in TC201.
    - **TC302: Multiple Remote Online Download**

      This test evaluates the download capacity of the target site for online data using it maximum concurrent download capacity. It is the same as TC301 with as many concurrent download as the configured maximum allows.

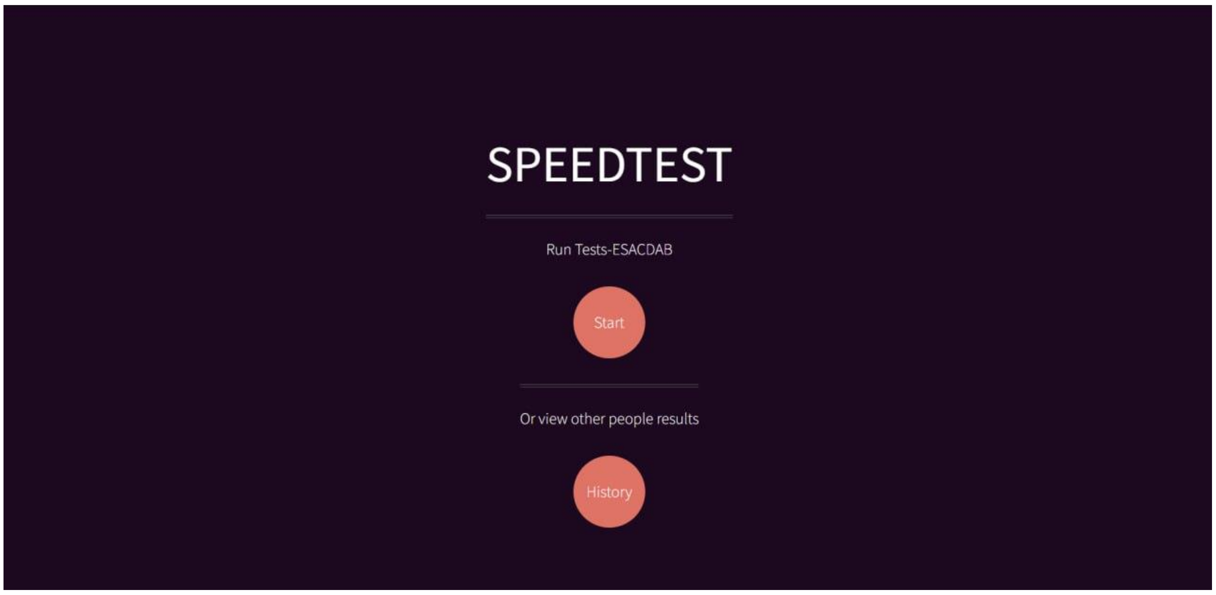
      The obtained metrics are the same as in TC301.
    - **Relevant Metrics:**
        - Average response time (M001)
        - Peak response time (M002)
        - Error rate (M003)

- And extensive metrics related to concurrency, data size, and error rates in results

10.1 Step-by-Step Execution Process

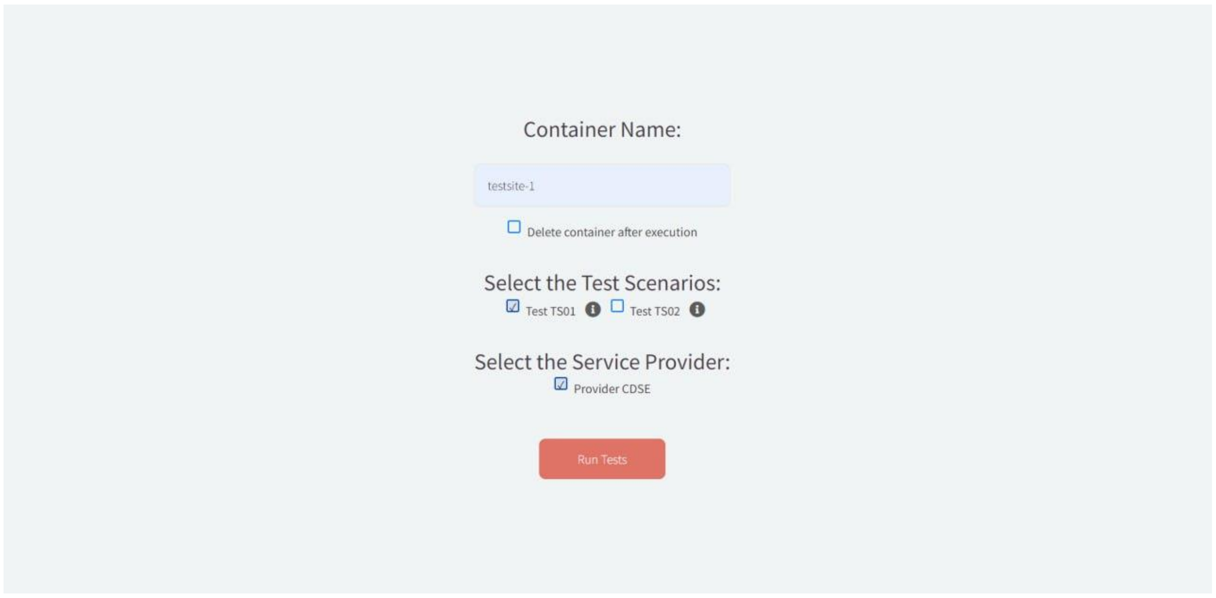
Starting the Test:

1. **Run Tests**: Begin by clicking the 'Start' button on the main dashboard to initiate the testing process.



2. **Setup Test Environment**:

- **Docker Container Name**: The name 'testsite-1' is preconfigured in the Python files, so it defaults in the input field. No user input is required here.
- **Select Test Scenarios**: Choose between TS01 (Simple Data Search and Single Download) or TS02 (Complex Data Search and Bulk Download), or both.
- **Select Service Provider**: Choose 'CDSE' as the provider, which stands for Copernicus Data Space Ecosystem.


Container Name:

testsite-1

☐ Delete container after execution

Select the Test Scenarios:

☑ Test TS01 ☐ Test TS02

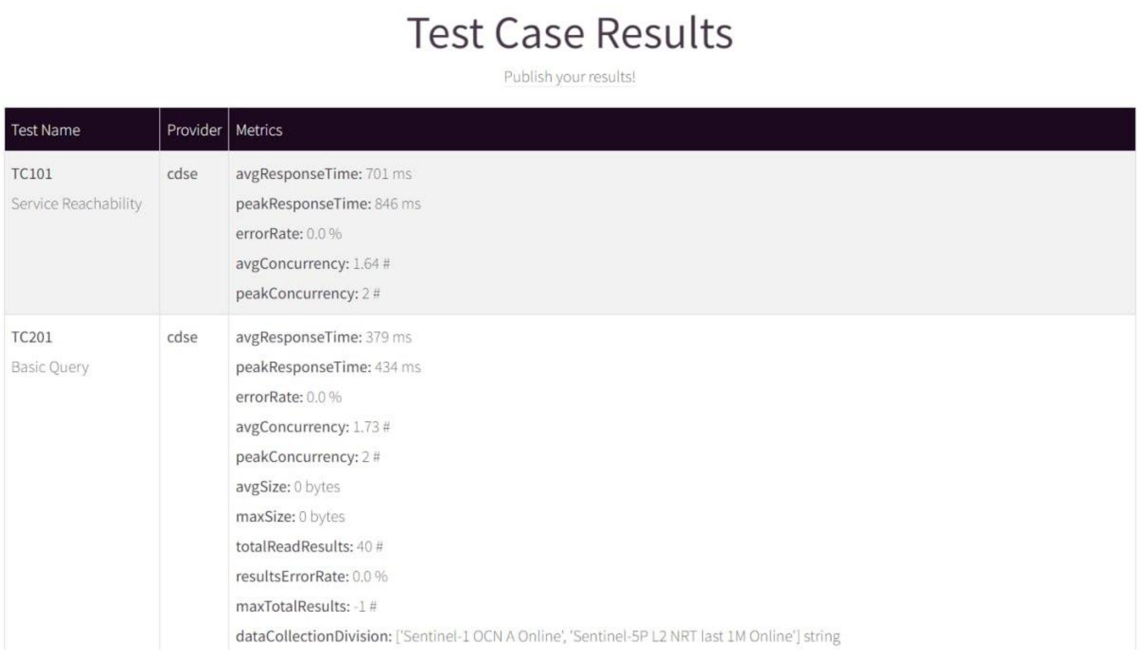
Select the Service Provider:

☑ Provider CDSE

3. **Run Execution**: After configuring the test settings, click 'Run Tests'. The system processes the request, performing the operations defined by the selected test scenarios.

Automated and Enhanced Visualization:

4. **Metrics Overview**:
    - After the test execution, a results page will display the collected metrics such as average response time, peak response time, error rate, and concurrency levels.
    - Click on 'Publish your results' to move to the next step

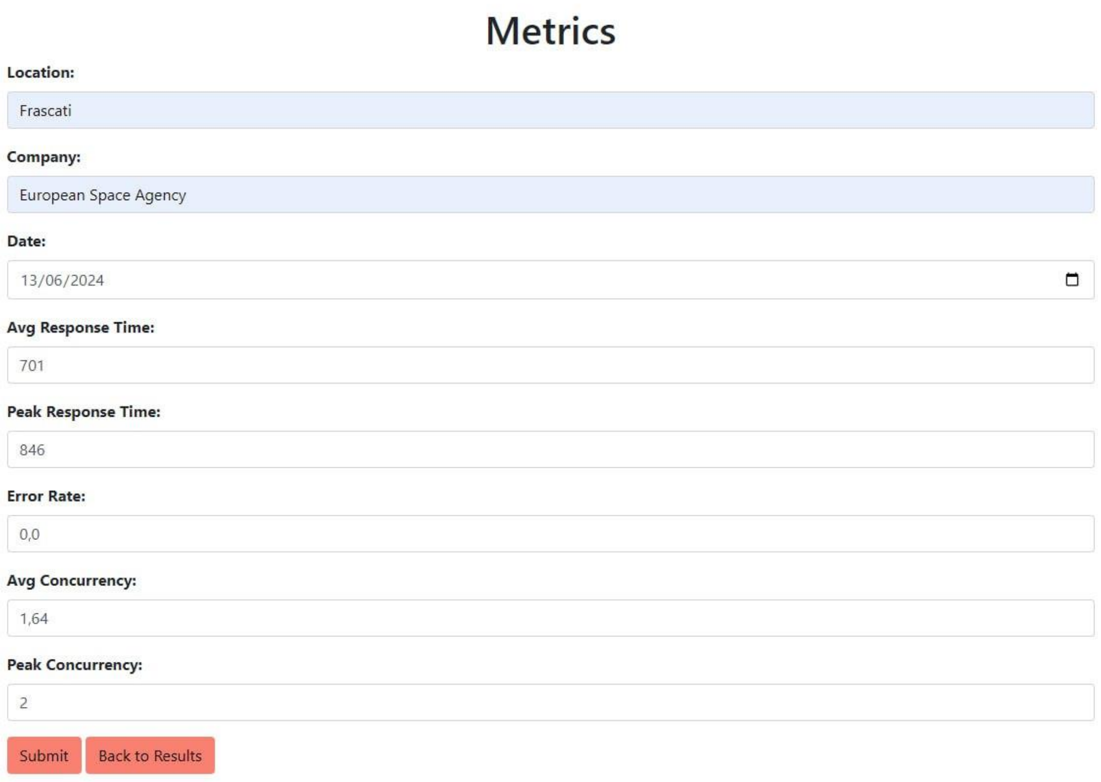


Test Case Results

Publish your results!

Test Name	Provider	Metrics
TC101 Service Reachability	cdse	**avgResponseTime:** 701 ms **peakResponseTime:** 846 ms **errorRate:** 0.0 % **avgConcurrency:** 1.64 # **peakConcurrency:** 2 #
TC201 Basic Query	cdse	**avgResponseTime:** 379 ms **peakResponseTime:** 434 ms **errorRate:** 0.0 % **avgConcurrency:** 1.73 # **peakConcurrency:** 2 # **avgSize:** 0 bytes **maxSize:** 0 bytes **totalReadResults:** 40 # **resultsErrorRate:** 0.0 % **maxTotalResults:** -1 # **dataCollectionDivision:** ['Sentinel-1 OCN A Online', 'Sentinel-5P L2 NRT last 1M Online'] string

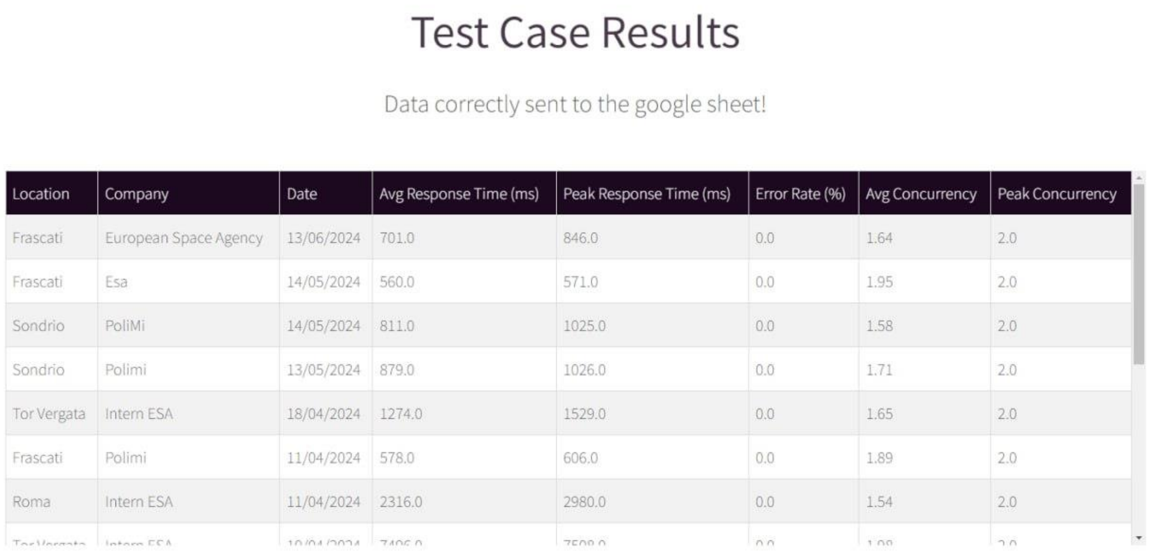
5. **Data Submission**:
   - You will see a form that auto-fills with the test metrics. Add details like the location and company name.
   - Submit the data to integrate with Google Sheets for record-keeping and further analysis.



Metrics

Location:	Frascati
Company:	European Space Agency
Date:	13/06/2024
Avg Response Time:	701
Peak Response Time:	846
Error Rate:	0,0
Avg Concurrency:	1,64
Peak Concurrency:	2

6. **Historical Data Review**:
   - Navigate directly from the results submission or access through the 'History' button on the main page.
   - This section displays all past results in a tabulated format and allows comparisons with other data points.



Test Case Results

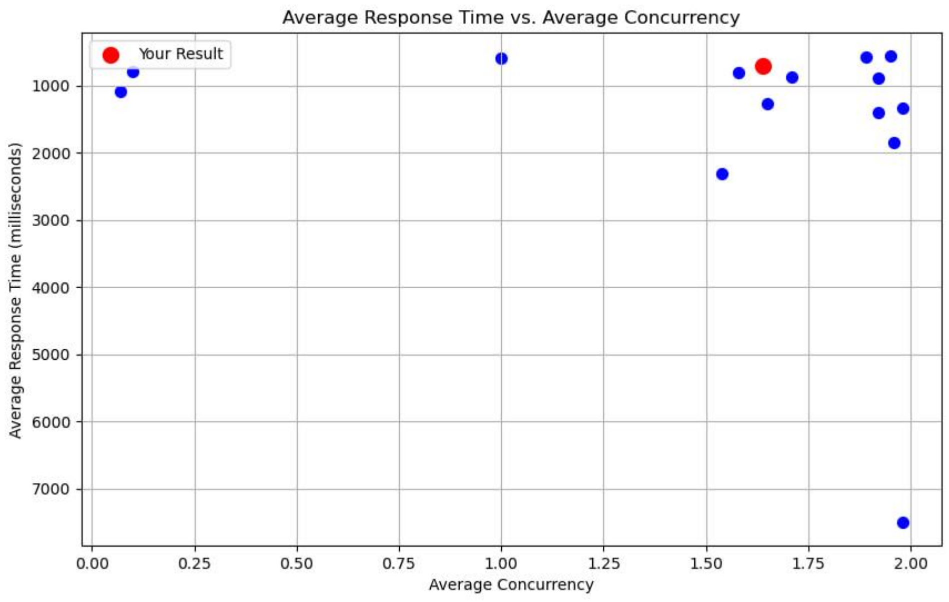
Data correctly sent to the google sheet!

Location	Company	Date	Avg Response Time (ms)	Peak Response Time (ms)	Error Rate (%)	Avg Concurrency	Peak Concurrency
Frascati	European Space Agency	13/06/2024	701.0	846.0	0.0	1.64	2.0
Frascati	Esa	14/05/2024	560.0	571.0	0.0	1.95	2.0
Sondrio	PoliMi	14/05/2024	811.0	1025.0	0.0	1.58	2.0
Sondrio	Polimi	13/05/2024	879.0	1026.0	0.0	1.71	2.0
Tor Vergata	Intern ESA	18/04/2024	1274.0	1529.0	0.0	1.65	2.0
Frascati	Polimi	11/04/2024	578.0	606.0	0.0	1.89	2.0
Roma	Intern ESA	11/04/2024	2316.0	2980.0	0.0	1.54	2.0
Tor Vergata	Intern ESA	10/04/2024	7406.0	7508.0	0.0	1.98	2.0

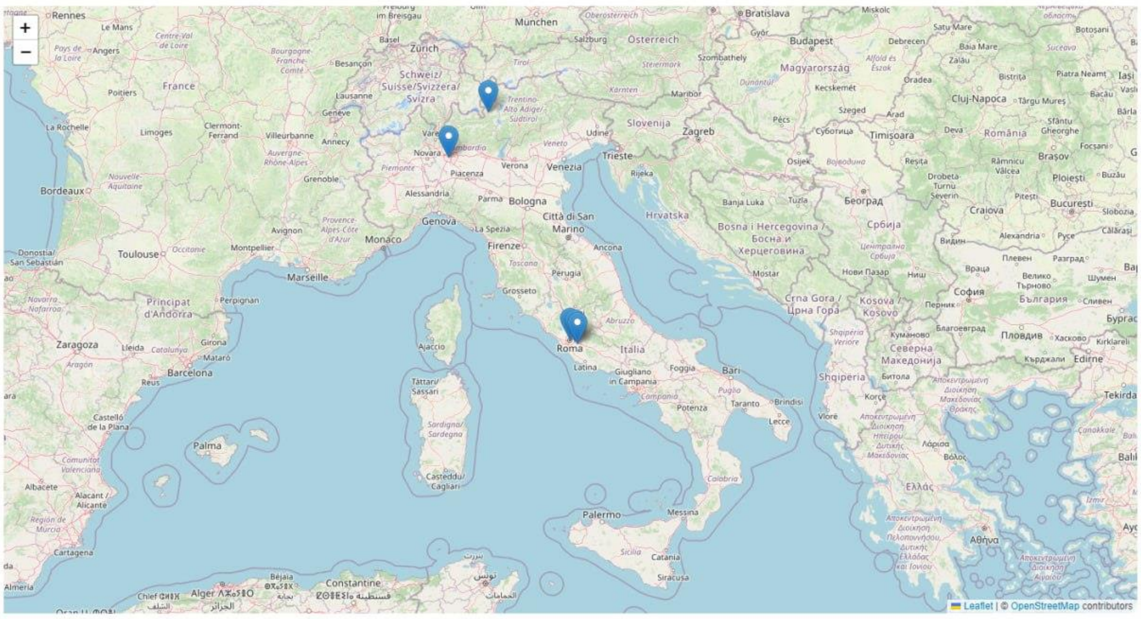
Interactive and Graphical Data Interpretation:

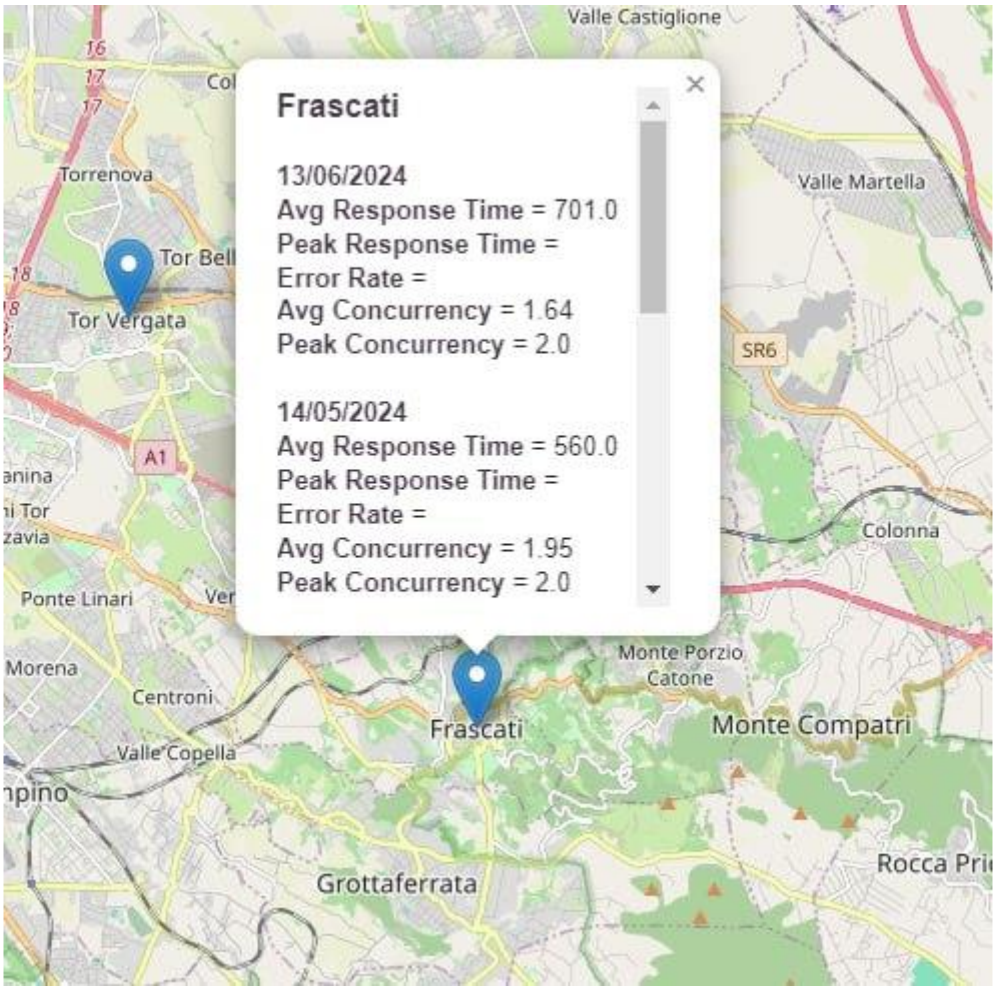
7. **Graphical Comparisons**:
    - View graphs that plot metrics like average response time against concurrency, allowing visual comparison of your results against others. The map is designed so that the higher you are to the top right, the better your performance is.

Results plot



8. **Interactive Map Feature**:
    - An interactive map showcases global test executions. Clicking on map markers reveals specific metrics for each location, enhancing the geographical analysis of data.





Frascati

13/06/2024
Avg Response Time = 701.0
Peak Response Time =
Error Rate =
Avg Concurrency = 1.64
Peak Concurrency = 2.0

14/05/2024
Avg Response Time = 560.0
Peak Response Time =
Error Rate =
Avg Concurrency = 1.95
Peak Concurrency = 2.0

11 Contributing to the Project

11.1 How to Contribute

The CDAB Simple Plugin is hosted on GitHub, making it an open-source project where contributions are highly encouraged and welcomed. Contributors can enhance the plugin in various ways, such as adding new features, improving existing functionalities, or expanding the test scenarios. Here's how to get involved:

- **Explore the Repository**: Start by familiarizing yourself with the project's structure and content at our GitHub repository: https://github.com/Virginia555/cdseesa-Plugin
- **Clone the Repository**: Clone the repository to your local machine to start working on your improvements.
- **Add New Plugins or Test Cases**: While the repository initially includes basic test scenarios for simplicity, you are encouraged to add new ones that might be relevant to different data providers.
- **Expand Provider Support**: The plugin currently supports CDSE as a target site, but additional providers can be added. Contributors can integrate more target sites by modifying the existing Python code to accommodate new test configurations.
- **Submit Pull Requests**: After making your changes, submit a pull request for review. Include a clear description of the modifications and their benefits to the project.
- **Documentation**: Help improve or update the documentation to ensure that all functionalities are well documented and easy to understand.

11.2 Feedback and Suggestions

Feedback is crucial for the continual improvement of the CDAB Simple Plugin. Whether it's a feature request, bug report, or general suggestions, your feedback helps enhance the tool's effectiveness and usability.

12 Conclusion

The CDAB Simple Plugin, as detailed in this report, exemplifies an advancement in the domain of benchmarking data providers using the Copernicus Sentinels Data Access Worldwide Benchmark Test Suite. By offering an intuitive and efficient web-based interface, the plugin mitigates the complexities associated with traditional benchmarking tools. It enhances user accessibility, supports diverse operating systems, and integrates advanced data visualization mechanisms, including automated Google Sheets and interactive maps. These features collectively streamline the benchmarking process, making it accessible to a broader audience and facilitating more reliable and timely data services. The plugin's development underscores the commitment to improving the user experience in data accessibility testing and aligns with the comprehensive goals of the Copernicus program to provide consistent and predictable data services globally. Future enhancements and continued stakeholder engagement will be pivotal in maintaining and expanding the utility of the CDAB Simple Plugin.

Bibliography

[1] Virginia555, "cdseesa-Plugin," GitHub Repository. Available online: https://github.com/Virginia555/cdseesa-Plugin

[2] ESA-CDAB, "Copernicus Data and Access Benchmark Testsuite," GitHub Repository. Available online: https://github.com/esacdab/cdab-testsuite

[3] ESA-CDAB, "Copernicus Data and Access Service," GitHub Repository. Available online: https://github.com/esa-cdab/cdab-service

[4] ESA-CDAB, "Service Design Document V2.2.1b," 2024. Available online: https://github.com/esa-cdab/cdab-service/blob/main/docs/Service_Design_Document_V2_2_1b_signed.pdf

A. Code Snippets

These snippets should provide clear examples of how the application works, particularly focusing on parts of the code that handle the core functionalities.

Running Tests and Handling Responses:

```
#Request when the form is submitted
@app.route('/run_tests', methods=['POST'])
def run_tests_route():

    #Retrieve the data from the request
    container_name = request.form['container_name'].strip()
    delete_after_execution = 'delete_container' in request.form
    tests = request.form.getlist('tests')
    providers = request.form.getlist('providers')
    validDockerName = re.compile(r'^[a-zA-Z0-9][a-zA-Z0-9_.-]*$') #Regex for
valid Docker container name

  #Run the tests (imported function from functions.py)
    results = run_tests(container_name=container_name, tests=tests,
providers=providers, remove_container=delete_after_execution)

    if results is False: #Result is set to false when there is something wrong
with Docker
        return redirect(url_for('error', type='docker'))

    #If we got to the results, render the template with the results and add
them to the session
    session['results'] = results
    return render_template('results.html', results=results)
```

Google Sheets Integration:

```
def write_to_sheet(data):
    # Carica le credenziali dal file del service account
    creds = Credentials.from_service_account_file(
        "credentials.json", scopes=SCOPES)

    # Costruisci il client per l'API di Google Sheets
    service = build("sheets", "v4", credentials=creds)

    try:
        # Chiama l'API di Google Sheets per scrivere i dati
        sheet = service.spreadsheets()
        # Calcola la prossima riga vuota nel foglio di lavoro
        next_row = len(sheet.values().get(spreadsheetId=SAMPLE_SPREADSHEET_ID,
                                          range=SAMPLE_RANGE_NAME).execute().
get("values", [])) + 1
        # Scrivi i dati nella prossima riga vuota del foglio di lavoro
        sheet.values().update(spreadsheetId=SAMPLE_SPREADSHEET_ID,
                              range=f"data!A{next_row}",
                              valueInputOption="USER_ENTERED",
                              body={"values": [data]}).execute()
    except HttpError as err:
        print(err)
    session["success"] = True
```

```
ss

@app.route('/metrics_form', methods=['GET', 'POST'])
def metrics_form():
    if request.method == 'POST':
        location = request.form['location']
        company = request.form['company']
        date = request.form['date']
        avg_response_time = request.form['avgResponseTime']
        peak_response_time = request.form['peakResponseTime']
        error_rate = request.form['errorRate']
        avg_concurrency = request.form['avgConcurrency']
        peak_concurrency = request.form['peakConcurrency']

        # Write data to Google Sheet
        data = [location, company, date, avg_response_time,
peak_response_time, error_rate, avg_concurrency, peak_concurrency]
        write_to_sheet(data)
        return redirect(url_for('history'))
```

Checking Docker Status:

```
def is_docker_running():
    try:
        client = from_env()
        client.ping()  # Attempts to ping the Docker daemon
        print("Docker is running.")
        return True
    except DockerException:
        print("Docker is not running.")
        return False
```

Checking if a Docker Container Exists:

```
def check_container_exists(name):
    client = from_env()
    container_list = client.containers.list(all=True)
    for container in container_list:
        if name == container.name:
            return True
    return False
```

Running Tests using Docker:

```
def run_tests(container_name = "testsite-1", tests = ["TS01"], providers =
["cdse"], remove_container = True):

    #If docker is not running the returned false will trigger on app.py a
redirection to /error/docker
    if not is_docker_running():
        return False

    results_files = []

    for provider in providers:

        if not check_container_exists(container_name):
            print(f"Creating container {container_name}...")
            run_command(f"docker run --detach --name {container_name}
ghcr.io/esacdab/cdab-testsuite:latest")
        else:
            print(f"Container {container_name} already exists. Continuing
without creating a new one.")
            run_command(f"docker start {container_name}")

        run_command(f"docker cp config.yaml
{container_name}:/home/jenkins/config.yaml")

        for test in tests:
            print(f"Executing {test} on {provider}...")
            run_command(f"docker exec {container_name} cdab-client -v -
tsn={container_name} -tn={provider} {test}")

            result_file = f"{container_name}-{test}-{provider}-results.json"
            results_dir = os.path.join(os.path.dirname(__file__), 'Results')
            os.makedirs(results_dir, exist_ok=True)  # Creates the directory
if it does not exist
            local_result_file = os.path.join(results_dir, result_file)
            print(f"Copying results for {test} from {provider}...")
            run_command(f"docker cp
{container_name}:/home/jenkins/{test}Results.json {local_result_file}")
            results_files.append(local_result_file)

    if remove_container:
        print(f"Stopping and removing container {container_name}...")
        run_command(f"docker stop {container_name}")
        run_command(f"docker rm {container_name}")
```

Generate results from JSON Files:

```
#Generate a results array from the json files to be rendered by the template
def generate_results(results_files):
    results = []

    for json_file in results_files:
        provider = json_file.split('-')[-2]
        if os.path.exists(json_file):
            with open(json_file, 'r') as file:
                data = json.load(file)
                for result in data.get('testCaseResults', []):
                    result['provider'] = provider
                    result['description'] = descriptions[result['testName']]
                    results.append(result)

    return results
```

Result Visualization:

```
@app.route('/history', methods=['GET', 'POST'])
def history():
    sheet_id = '1kPkhk67xB-buh6V4EsR9-M2-cub_SreJ56ivTYjCRgk'
    df =
pd.read_csv(f'https://docs.google.com/spreadsheets/d/{sheet_id}/export?format=
csv')
    results = df.to_dict(orient='records')
    #session["success"] = True #For debugging purposes, don't uncomment it
    if session.get('success', False):
        session["success"] = False
        row_to_highlight = len(df)-1
        plot_scatter(df, row_to_highlight)
        return render_template('history.html', results=results, success=True)
    plot_scatter(df=df)
    return render_template('history.html', results=results, success=False)
```

```
def plot_scatter(df, row_to_highlight=False):
    plt.figure(figsize=(10, 6))
    plt.title('Average Response Time vs. Average Concurrency')
    plt.xlabel('Average Concurrency')
    plt.ylabel('Average Response Time (milliseconds)')
    plt.grid(True)

        # Highlight a specific row if indicated
    if row_to_highlight:
        plt.scatter(df.drop(row_to_highlight)['avgConcurrency'],
df.drop(row_to_highlight)['avgResponseTime'], color='blue', s=50)
        my_point = plt.scatter(df.loc[row_to_highlight, 'avgConcurrency'],
df.loc[row_to_highlight, 'avgResponseTime'], color='red', s=100, label='Your
Result')  # Larger and red dot
        plt.legend(handles=[my_point])
    else:
        plt.scatter(df['avgConcurrency'], df['avgResponseTime'], color='blue',
s=50)  # s is the size of the dots

    plt.gca().invert_yaxis()
```