IBIMA Publishing
Communications of International Proceedings
https://ibimapublishing.com/p-articles/46SE/2025/4622425/
Vol. 2025 (36), Article ID 4622425

Integrated, Containerized JupyterHub Environment for a Higher Education Institution: A Case Study of Deployment*

Mariusz BORKOWSKI, Anna KOCAN-KRAWCZYK and Pawel WLADYKA

Faculty of Electrical and Computer Engineering, Rzeszow University of Technology, Rzeszow, Poland,

Correspondence should be addressed to: Mariusz BORKOWSKI, marbor@prz.edu.pl

* Presented at the 46th IBIMA International Conference, 26-27 November 2025, Ronda, Spain

Abstract

The article presents a case study of deploying the JupyterHub platform at Rzeszów University of Technology for teaching purposes. The system was designed as a modular container-based environment built on Docker and Docker Compose, fully integrated with the university's Central Authentication Service (CAS). The architecture provides secure, isolated programming environments, differentiation of user roles (students and teachers), and mechanisms for controlled distribution of teaching materials.

The paper discusses key engineering challenges related to adapting open-source tools to the specific requirements of an academic institution, including authentication integration, user role management, and automated data backups. Performance tests were conducted to evaluate the stability and scalability of the environment under varying workloads.

The implemented solution, available to authenticated users at https://jupyterhub.prz.edu.pl, demonstrates that lightweight containerization technologies can successfully support the teaching process in higher education. The project also provides practical insights for other institutions seeking to implement scalable, maintainable, and secure notebook-based environments.

Keywords: JupyterHub, Docker, containerization, higher education, e-learning infrastructure.

Introduction

In recent years, the growing demand for remote learning formats and the ongoing digitalization of academic instruction have driven the rapid development of platforms supporting online and hybrid teaching, AlDuhisat et al. (2024); Ochkov et al. (2022). The COVID-19 pandemic further accelerated this process, exposing the limitations of traditional university infrastructure and highlighting the need for flexible and easily scalable access to programming environments, Sagale et al. (2023).

Cite this Article as: Mariusz BORKOWSKI, Anna KOCAN-KRAWCZYK and Pawel WLADYKA, Vol. 2025 (36) "Integrated, Containerized JupyterHub Environment for a Higher Education Institution: A Case Study of Deployment " Communications of International Proceedings, Vol. 2025 (36), Article ID 4622425, https://doi.org/10.5171/2025.4622425

In response to these needs, many academic institutions have turned to open-source solutions that enable the execution of computational environments directly through the web browser. At the forefront of such tools is the JupyterHub platform (Jupyter Team (2025)), which allows multiple users to run individual notebook environments in parallel, with full session isolation and support for a wide range of programming languages (Python, R, Julia, C++).

Over the past years, numerous variants and extensions of JupyterHub have been developed, tailored to the specific requirements of different deployment scenarios. One such solution is The Littlest JupyterHub (TLJH) Jupyter Project (2022) - a simplified distribution designed for quick, local deployments on a single server. TLJH is targeted at users who may not have extensive administrative expertise but wish to rapidly provide a Jupyter notebook environment for their students or collaborators. JupyterHub on Kubernetes, by contrast, is a highly scalable solution that integrates the platform with cloud infrastructures or container clusters, Sarajlic et al. (2018). Another example is OpenDreamKit, an initiative aimed at delivering a suite of tools (including JupyterHub) to support scientific collaboration and teaching in mathematics and the natural sciences, OpenDreamKit Consortium (2019).

Each of these solutions provides distinct benefits but also has inherent limitations. TLJH, while quick to set up, lacks advanced resource management features and does not natively support integration with external authentication systems. On the other hand, Kubernetes offers superior scalability but is complex and challenging to maintain for teams without prior experience in container orchestration. OpenDreamKit proposes a rich and flexible environment but is primarily oriented toward research collaboration rather than course-centered teaching.

The deployment presented in this article represents a compromise between flexibility and complexity. It is based on containerization of JupyterHub using Docker and Docker Compose, with the primary goal of supporting programming courses in multiple languages while ensuring full integration with the university's Central Authentication Service (CAS). The key technical objectives of the project include:

- building an infrastructure capable of supporting hundreds of concurrent users,
- full integration with the institutional identity management system (CAS + user roles),
- secure and controlled management of shared teaching materials,
- and ensuring data resilience through container volume backups.

Compared to existing solutions, the project stands out due to:

- the use of a lightweight Docker-based architecture instead of Kubernetes,
- the adoption of Traefik as a dynamic reverse proxy and load balancer,
- an extended mechanism for managing public and private folders according to user roles,
- and a detailed performance testing methodology utilizing Selenium.

The aim of this article is not only to present the implemented solution but also to identify the engineering challenges encountered during integration with the university's IT infrastructure and to discuss the strategies employed to overcome them.

The article takes the form of a case study and is intended for academic system administrators, IT educators, and individuals interested in the use of JupyterHub within the real-world context of higher education institutions.

The following sections present the technical background of the project, the integration challenges, the system architecture, as well as the testing methodology and the performance analysis of the environment.

System architecture and implementation

The system deployed for the teaching needs of Rzeszów University of Technology was designed as a modular application based on Docker container technology, managed with Docker Compose. The goal of the architecture was to provide an isolated, secure, and easy-to-manage programming environment, fully integrated with the university's CAS authentication infrastructure. The main components of the system include:

- JupyterHub – the component managing user sessions and launching JupyterLab containers,
- DockerSpawner – an extension that enables the execution of individual user environments (JupyterLab) as containers,
- Traefik – the reverse proxy responsible for request routing, TLS handling, and dynamic HTTP/HTTPS redirections,
- Docker Compose – the orchestration tool used to manage all system services.

These modules and the interactions between them are illustrated in Figure 2.1. The complete system configuration – including container definitions, Dockerfiles, JupyterHub and Traefik configuration files, as well as auxiliary scripts (backup, synchronization of shared folders) – has been made publicly available in the GitHub repository (Borkowski (2025)).

The project structure largely corresponds to the system components list presented above and includes, among others:

- directory jupyterhub/ containing the JupyterHub configuration, including the integration logic with CAS,
- directory notebook/ defining the user's JupyterLab environment image (with selected kernels),
- directory proxy/ with configuration files and the Traefik container definition,
- file docker-compose.yml aggregating all components and data volumes.

Authentication -- CAS integration

To ensure single sign-on (SSO), the component jhub_cas_authenticator is used. It enables communication between JupyterHub and the CAS server. After the user is redirected to the CAS login page and successfully authenticated, a set of user attributes (so-called CAS attributes) is sent to JupyterHub. Among them, the email address is of key importance, as it allows distinguishing user role. Two types of roles are distinguished: student and teacher.

The role is assigned to the user internally in the jupyterhub_config.py configuration and determines:

- the assignment of the appropriate home folder,
- access to shared directories,
- the permission to publish educational materials.

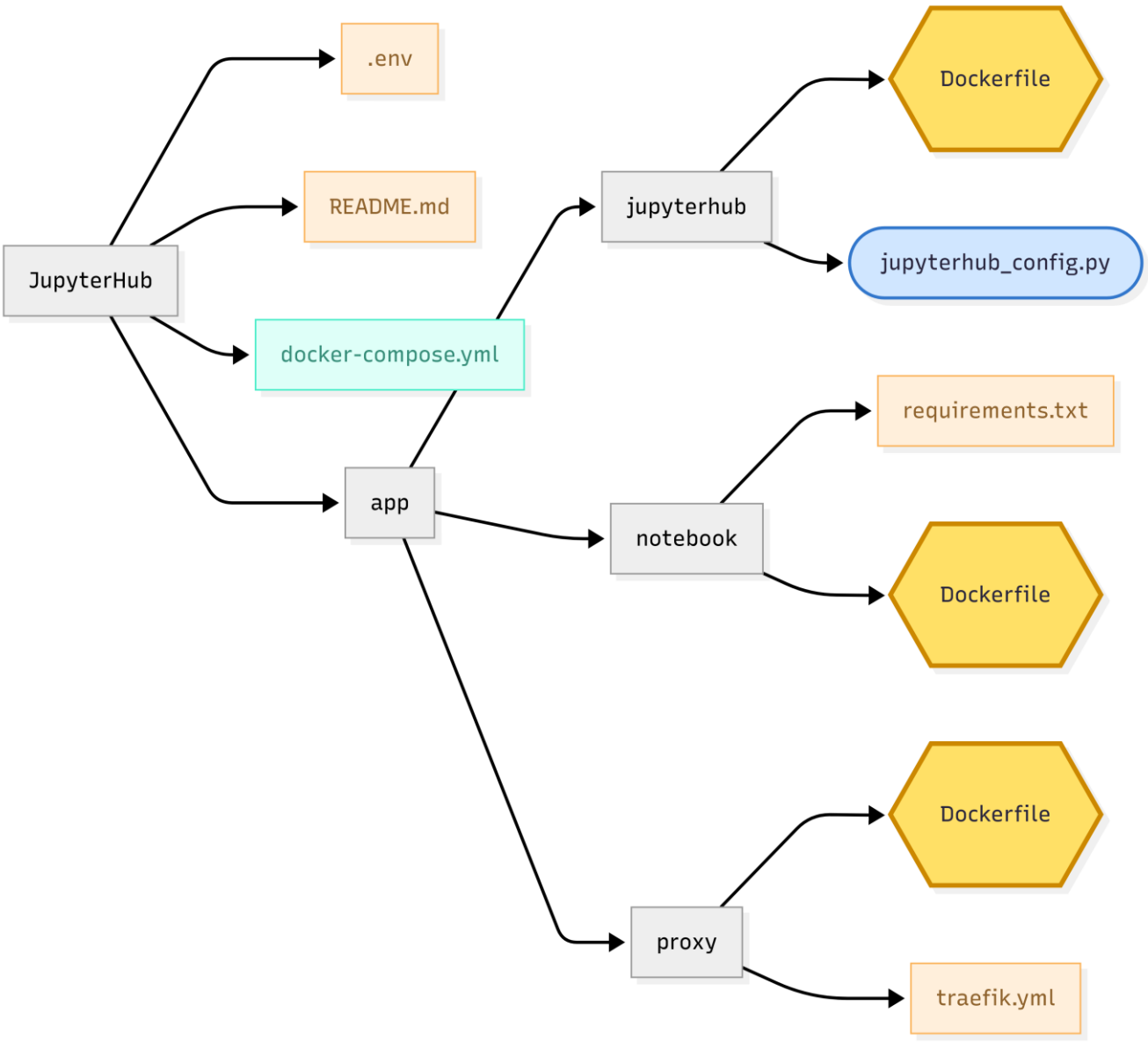

Fig 1. System architecture diagram

Resource and session management

The JupyterHub environment has been designed to allow multiple users to share server resources while ensuring environment isolation and limiting the potential for excessive resource consumption by individual sessions. To this end, the following resource management mechanisms have been implemented:

- mem_limit and cpu_limit in the DockerSpawner configuration: memory and CPU usage limits are set for each user container,
- active_server_limit: restriction on the number of simultaneously running containers, preventing system overload during high concurrency logins,

- c.Spawner.default_url: setting the default environment launched after login (JupyterLab instead of the classic Notebook),
- jupyterhub-idle-culler: an extension that automatically closes sessions that have been inactive for a defined period to reclaim resources.

From the user's perspective, all these mechanisms are invisible, yet they significantly enhance overall system stability. Additionally, each user has a persistent volume (work) where notebook data, scripts, and files are stored. These volumes are mounted every time a session is started, ensuring data persistence between logins – even if the user's container is deleted or restarted.

Deployment and automation

The entire system has been designed so that its deployment and maintenance do not require advanced DevOps knowledge. Thanks to the use of docker-compose, all components – JupyterHub, Traefik, user container, data volumes – are defined in a single file (docker-compose.yml), and starting the system requires only a single command: docker-compose up -d.

Additionally, the project repository (Borkowski (2025)) contains scripts facilitating the automation of administrative processes:

- backup_script.sh – creates a backup of user directories and system data, including work and public volumes,
- sync_jupyter_public.sh – synchronizes educational materials submitted by teachers (from my_public/ directory) to the shared public/ directory,
- test_environment.sh – enables quick startup of a test environment with DummyAuthenticator for testing new features without interacting with CAS.

The separation of startup logic (production and test), modular configuration files, and clear distinction of user roles make the system easily replicable, portable, and capable of being further developed by other technical teams or university units.

Data volumes and their role

The deployment assumes the existence of four types of volumes, defined in docker-compose.yml:

- work – the user's private environment,
- public – a shared directory with educational materials (read-only for students),
- my_public – an intermediate layer allowing data publication while maintaining file system permission constraints,
- readme_dir – a directory containing documentation and startup instructions.

Detailed scripts, configuration parameters, and deployment documentation are provided in the project repository, Borkowski (2025).

Educational materials management

One of the key aspects of the deployment was the development of a model for managing educational materials that would ensure:

- the ability for teachers to share content,
- secure, read-only access to these materials for students,
- flexibility in updating and synchronizing data by system administrators.

Since the environment is based on Docker containers dynamically launched for each individual user, managing the folder structure and assigning permissions had to take into account the constraints resulting from the file system isolation of each container, default Linux permission settings (UID/GID), and differences in user roles (student vs teacher).

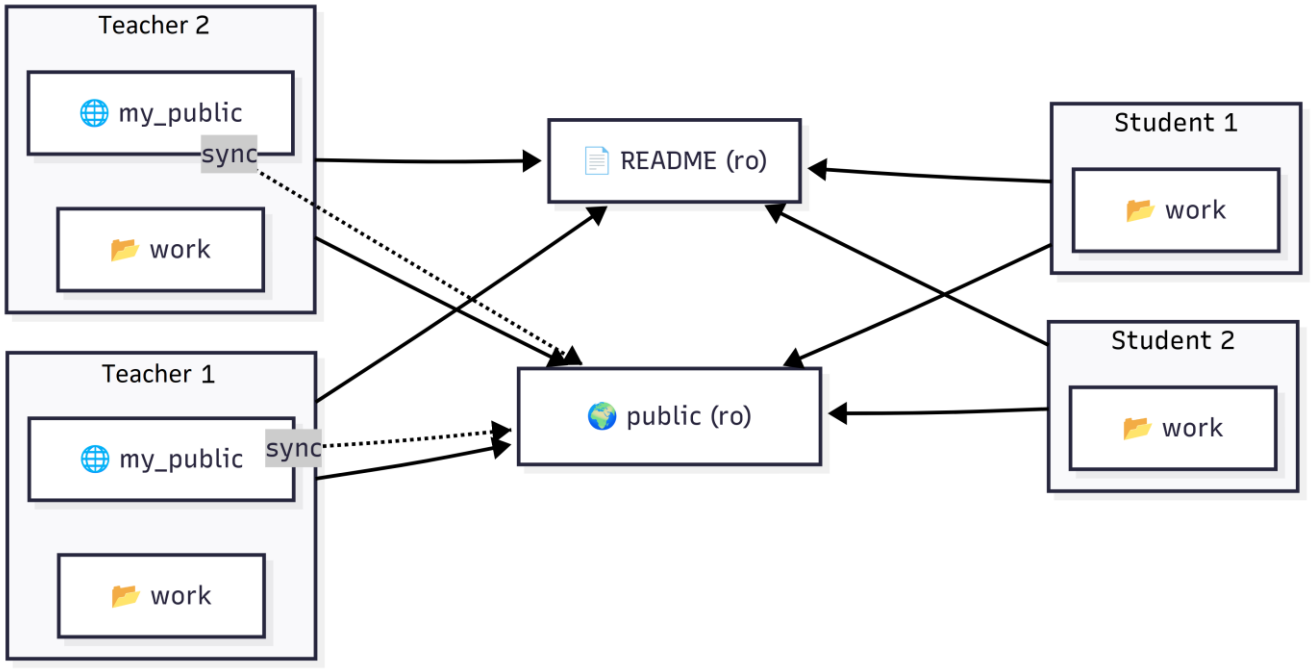

Fig 2. Diagram of Docker volume management organization in the presented system

Directory structure and data volumes

The system defines four mount points for volumes:

- work/ – the individual directory of each user, serving as the main working environment,
- public/ – a shared directory from which all users can read materials,
- my_public/ – a directory enabling teachers to publish files; mounted only for accounts with the teacher role,
- readme_dir/ – a directory with technical instructions and informational materials, available in read-only mode to every user.

Upon login, a student is granted access only to the directories: work/, public/, and readme_dir/, while a teacher additionally receives the my_public/ volume, where new educational materials can be placed. Then, through an external synchronization script (sync_jupyter_public.sh), files from my_public/ are copied to the public/ directory with read-only permissions preserved for other users.

This approach provides a workaround for Docker's lack of support for dynamically assigning permissions to shared volumes in a multi-container environment, where each container operates with different UIDs.

Separation of user roles

In the IT infrastructure of the described university system, the following assumption was adopted: student account logins consist of a unique six-digit identification number with the postfix @stud.prz.edu.pl (e.g., 123456@stud.prz.edu.pl). In contrast, staff account logins are alphanumeric strings, usually an abbreviation of the first and last name, ending with the postfix @prz.edu.pl (e.g., alkow@prz.edu.pl). This rule makes it possible to determine the user's role in several ways. In the presented solution, the decision was made to check whether the login prefix is a six-digit string. If so, the logging user is recognized as a student; otherwise, the user is assumed to be a teacher.

At the time of login via CAS, JupyterHub assigns the user the role student or teacher. Based on this information, selective mounting of volumes occurs (my_public only for teachers), along with the choice of the start directory and accessible resources, as well as the definition of the interface and available functions in JupyterLab.

The entire logic for role assignment and directory mounting is contained in the jupyterhub_config.py file, available in the project repository, Borkowski (2025).

Content Management

Each teacher is responsible for updating educational materials by placing new files or directories in the my_public/ directory. The contents of these directories are automatically synchronized with the public/ directory, where the my_public/ directory of user Teacher-A is visible under the name public/Teacher-A/. This operation is carried out through the automated execution of the sync_jupyter_public.sh script. The script handles overwriting existing files, updating metadata, and logging operations (e.g., synchronization dates). As a result, the publication process is transparent, and users with the assigned role student cannot interfere with the content published by teachers.

In summary, the system for managing educational materials was a key element of the deployment and, in the authors' view, the greatest conceptual challenge. Thanks to the modular approach and role separation, it provides both ease of use and full administrative control.

Performance testing methodology

To evaluate the stability and scalability of the deployed JupyterHub environment, a series of performance tests were conducted under conditions simulating real-world usage of the platform by students and teachers. The primary goals of the tests were to identify potential system bottlenecks, assess system response to increasing numbers of users, and verify the correctness of key functions (authorization, container startup, data backup).

Specifically, the tests aimed to:

- measure system response time during concurrent logins and notebook launches,
- assess system resource usage (CPU, RAM, I/O) under increasing load,
- verify the correct functioning of backups and folder synchronization during system operation,

- check proper session management and container stability.

To simulate user interactions with the JupyterHub platform, the Selenium library (Selenium Team (2025)) was used – a browser automation tool applied to simulate user logins via CAS and the launching of JupyterLab sessions. Auxiliary scripts developed for testing are available in the project repository (Borkowski (2025)) in tests/ directory.

Due to infrastructure limitations at the time of the study, the tests were conducted in an environment with limited computational resources. The obtained results should therefore be interpreted with respect to the capabilities of this specific configuration.

Despite these limitations, the experiments provided valuable insights into the efficiency of the deployed solution, its behavior under load, and potential directions for further optimization.

The tests were carried out in a test environment configured analogously to the production setup, using DummyAuthenticator instead of CAS. This enabled faster execution of repeatable tests with test accounts (student1, student2, …, studentN).

Three main test scenarios were defined:

1. Concurrent login and session startup

- Simulation of 10–50 users logging in within short intervals,
- Measurement of the time required for a complete environment startup (from login to notebook appearance),
- Recording CPU and memory load.

2. Maintaining active sessions and code execution

- In each session, the user executed simple code (e.g., matrix computations in NumPy, a for loop),
- Monitoring long-term resource utilization,
- Observing container stability (whether they crash or restart).

3. Data backup and material synchronization

- Running the backup script during active system usage,
- Verifying that the backup process did not freeze other Docker processes or negatively affect service availability,
- Measuring backup execution time under varying numbers of active users.

Additionally, in each scenario, errors were recorded in JupyterHub and Traefik logs, such as:

- incorrect redirects,
- timeouts during container startup,
- volume mounting errors.

Graphical representations of the test data for two sample configurations – 3 active and 10 passive users, and 3 active and 15 passive users – are presented in Figures 1 and 2, respectively.

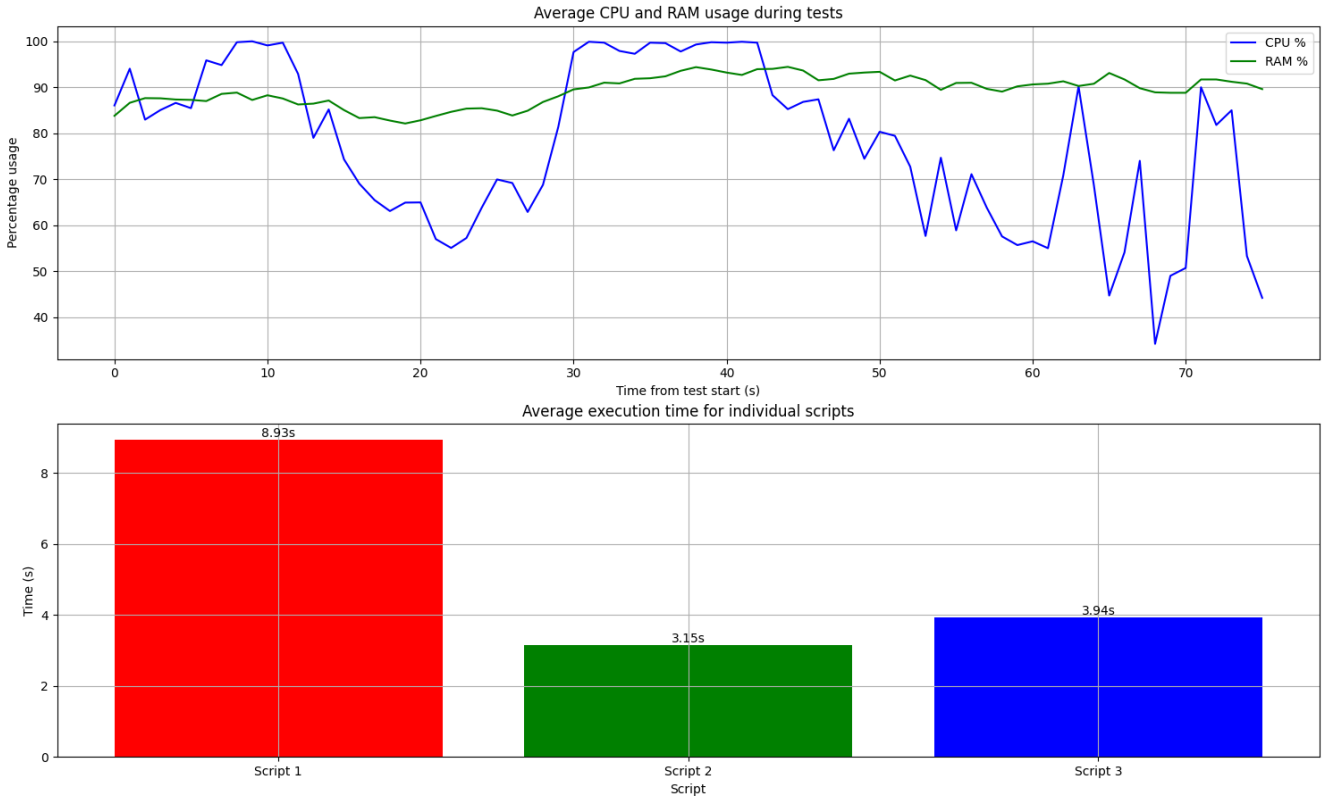

Fig 3. Average CPU and RAM utilization and average code execution time for 3 active and 10 passive users

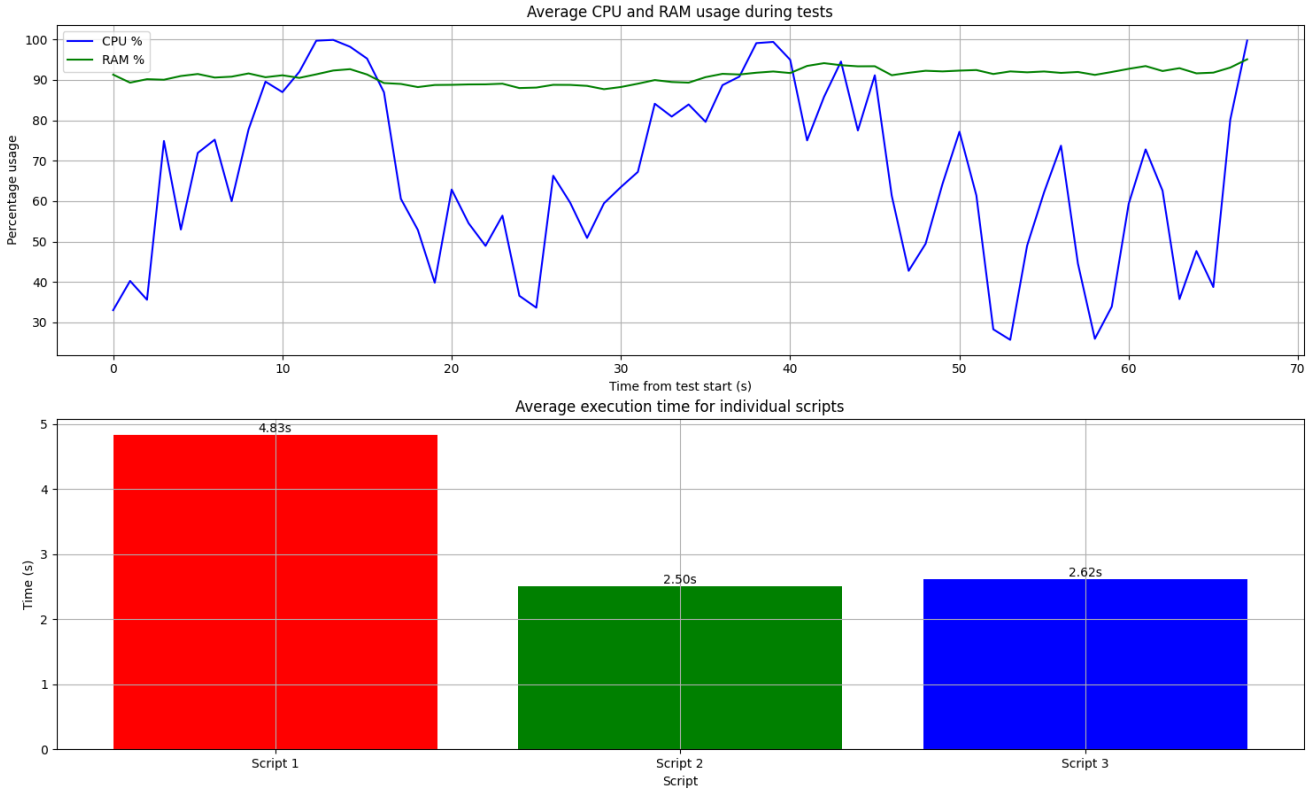

Fig 4. Average CPU and RAM utilization and average code execution time for 3 active and 15 passive users

Test results and analysis

Analysis of average CPU and RAM utilization indicates that an increase in the number of active users directly translates into more intensive CPU load. RAM usage remains at a relatively constant, high level throughout the entire experiment.

The growing number of simultaneously launched notebooks leads to significant CPU load, which remains elevated during the execution of computational codes.

Code execution times

Comparison of average task execution times shows a proportional increase as system load grows, both from active and passive users. The most time-consuming task turned out to be Code 1, which is consistent with its higher complexity. It is worth noting that mixed tests also confirmed the relationship between the number of users and system response time, even in situations where not all users performed intensive computations.

Load related to login and container creation

During load tests with 30 and 45 users, automated simulations were no longer used; instead, manual observations were made, including login times and overall platform responsiveness. It was found that the key factor affecting system responsiveness is the login process and initialization of new containers.

In the scenario with 30 users, an initial attempt to log in all participants simultaneously resulted in significant delays and temporary system freezes. After applying a gradual login strategy, the system functioned correctly without major issues, although the server's most demanding stage remained container creation. Using notebooks and running code no longer generated comparable load.

Scalability for larger numbers of users

In the scenario with 30 users (including 3 active and the rest passive), the platform operated efficiently and responsively, provided that simultaneous logins were avoided. In the 45-user test, the system remained functional but showed a noticeable deterioration in responsiveness and longer response times, particularly during container initialization.

Performance summary

In the test configuration, the JupyterHub platform was capable of smoothly handling around 30 concurrent users, provided they did not all run complex computations simultaneously. However, it should be emphasized that the login and container initialization stage represents the most demanding phase of system operation and may cause temporary delays.

At loads of around 45 users, responsiveness was already at the limit of acceptability, and operations such as login or file creation required noticeably more time. These findings indicate that when deploying the service for larger groups of simultaneously working users, increasing server resources will be necessary to maintain adequate performance quality.

The choice of kernel (Python, R, Octave) did not have a noticeable impact on user interface delays, demonstrating efficient resource management by the container mechanism.

Summary

This article presented a case study of the deployment of the JupyterHub platform at Rzeszów University of Technology, based on Docker and Docker Compose container technology and integrated with the university's CAS authentication system. The proposed solution represents a compromise between simplicity and flexibility on one hand, and the requirements for scalability and security on the other. The system is currently available to logged-in users at https://jupyterhub.prz.edu.pl.

A key achievement of the project was the development of an architecture that distinguishes user roles (student vs teacher), as well as the creation of a model for managing educational materials based on Docker volumes and dedicated synchronization scripts. This solution ensured both ease of use of the platform and full administrative control over educational content.

The conducted performance tests enabled an evaluation of system stability and its behavior under varying loads. They showed that the platform, in its current configuration, can effectively handle around 30 concurrent users, although the login and container initialization process remains the most critical stage of system operation. Further increases in the number of users would require expanding server resources or adopting more advanced orchestration strategies.

The conclusions from the deployment and testing confirm that open-source container technologies can successfully support academic teaching, providing a secure, scalable, and easy-to-manage computational environment. The project serves as a practical example of how open-source tools can be adapted to the specific needs of higher education institutions and may serve as a reference point for similar initiatives in other academic centers.

References

- Sabri, EH and Beamon, M. (2000), 'A Multi-Objective Approach to Simultaneous Strategic and Operational Planning in Supply Chain Design,' *Omega: an International Journal of Management Science* 28 (1), 581-598.
- AlDuhisat, M.H., AlKhresha, H.G., Al-Dmour, N.A., Ateeq, K., Maaitah, O.N., (2024). Role of cloud computing in education, in: 2024 2nd International Conference on Cyber Resilience (ICCR), pp. 1–4. doi:10.1109/ICCR61006.2024.10532895.
- Borkowski, M., (2025). PRzJupyterHub source code. URL: https://github.com/przemarbor/jupyterhub-docker/.
- Jupyter Project, (2022). The Littlest JupyterHub. URL: https://tljh.jupyter.org.
- Jupyter Team, (2025). Project Jupyter Documentation. URL: https://docs.jupyter.org/en/latest/.
- Ochkov, V.F., Stevens, A., Tikhonov, A.I., (2022). Jupyter notebook, jupyterlab – integrated environment for stem education, in: 2022 VI International Conference on Information Technologies in Engineering Education (Inforino), pp. 1–5. doi:10.1109/Inforino53888.2022.9782924.
- OpenDreamKit Consortium, (2019). Opendreamkit project. URL:https://opendreamkit.org.
- Sagale, K.S., Kokate, M.D., Agrawal, R.K., (2023). Application of cloud computing in an education sector through education and learning as a service and its cost benefit analysis, in: 2023 International Conference on Emerging Smart Computing and Informatics (ESCI), pp. 1–5. doi:10.1109/ESCI56872.2023.10099586.
- Sarajlic, S., Chastang, J., Marru, S., Fischer, J., Lowe, M., (2018). Scaling jupyterhub using kubernetes on jetstream cloud: Platform as a service for research and educational initiatives in the atmospheric sciences, pp. 1–4. doi:10.1145/3219104.3229249.
- Selenium Team, (2025). Selenium webpage. URL: https://www.selenium.dev.