A Cloud-Native Architecture for the Machine Teaching Platform: Enhancing Performance and Scalability

Nícolas da Mota Arruda[1], **Marcos T. Leipnitz**[1], **Laura O. Moraes**[2], **Carla Delgado**[1]

[1] Instituto de Computação – Universidade Federal do Rio de Janeiro (UFRJ)

[2]Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

{nicolasma,marcosleipnitz,carla}@ic.ufrj.br, laura@uniriotec.br

Abstract. *This article presents the redesign of the Machine Teaching platform from a browser-based online judge to a cloud-native architecture, offloading code execution to isolated containers that scale elastically via a serverless container service. This shift improves equitable access by reducing client-side hardware demands, strengthens security through server-side vulnerability analysis, and enables multi-language support without exposing evaluation logic in the browser. Experiments indicate faster responses on low-end devices and stable latency under concurrent load.*

Resumo. *Este artigo apresenta o redesenho da plataforma Machine Teaching de um juiz online baseado em navegador para uma arquitetura cloud-native, deslocando a execução de código para contêineres isolados que escalam elasticamente por meio de um serviço serverless de contêineres. Essa mudança melhora a equidade de acesso ao reduzir as exigências de hardware no cliente, reforça a segurança com análise de vulnerabilidades no servidor e viabiliza suporte multilinguagem sem expor a lógica de avaliação no navegador. Os experimentos indicam respostas mais rápidas em dispositivos de baixo desempenho e latência estável sob carga concorrente.*

1. Introduction

Platforms for programming education increasingly rely on secure, scalable execution of student-submitted code without imposing client-side hardware constraints. Originally, the Machine Teaching[1] (MT) platform [Moraes et al. 2022] evaluated code in the browser using Skulpt[2]/JS, which constrained security and performance on low-end devices. We redesigned MT as a cloud-native, container-based system that offloads code execution to isolated Worker Nodes, improving security, observability, and maintainability. The web application and workers run as independent serverless containers on Google Cloud Run[3], with horizontal autoscaling and per-instance concurrency limits. Moreover, a new security layer inside the containerized worker adds static code analysis, import/syscall filtering, and server-side timeouts, while enabling multi-language support without exposing evaluation logic in the browser.

The key contributions are: (i) the redesign of the MT platform into a cloud-native, containerized architecture supporting elastic scaling; (ii) an integrated security pipeline

[1]https://machineteaching.tech/

[2]https://skulpt.org/

[3]https://cloud.google.com/run/docs/overview/what-is-cloud-run

(pre-checks, timeouts, filters) for untrusted code; and (iii) empirical evidence of performance gains and scalability under load, improving the experience on low-end devices.

2. Related Work

Online judge systems are widely used in programming education and contests, providing fast feedback and consistent assessment of solutions [Carvalho et al. 2016]. The literature identifies security (for third-party code) and scalability (for submission spikes) as primary requirements, with modularity, load balancing, and isolated execution underpinning robustness [Watanobe et al. 2022]. In Brazil, immediate-feedback tools are well received among students and support active learning methodologies. The MT platform exemplifies this by coupling online judging with educational data for pedagogical decisions [Moraes et al. 2022]. However, in-browser execution raises security risks and performs poorly on modest devices, hindering equitable access. Therefore, a cloud-native, containerized solution is a promising option to mitigate these issues by isolating code submissions and improving portability [Wasik et al. 2018].

3. Cloud-Native Architecture

3.1. Components and Deployment

Figure 1 shows the new MT architecture on Google Cloud Platform. The Web App and the Worker Node run as isolated, stateless containers on Google Cloud Run, benefiting from elastic autoscaling to handle load spikes. A dedicated VM on Google Compute Engine runs a PostgreSQL[4] database to persist user accounts, problem catalog, and submission/grading records. This separation supports low-end devices by offloading the processing workload to dedicated servers while keeping state consistent and recoverable.

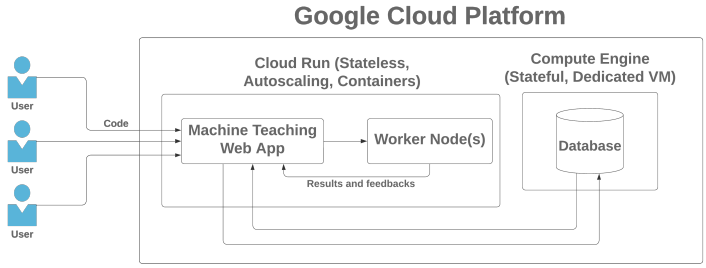

Figure 1. System architecture on the Google Cloud Platform

The Worker Node receives jobs from the Web App and executes them in sandboxed environments with CPU/memory quotas and wall-clock timeouts, producing verdicts. We use the Python subprocess library[5] to spawn external processes, execute code in different languages, and capture their output. A modular layer ensures extensibility and eases adding other languages, which allowed MT to be expanded to support C and Julia.

3.2. Execution Pipeline

A submission is processed through a multi-step execution pipeline. First, the Web App submits the student's code to the Worker Node. Next, the code undergoes a pre-processing

[4]https://www.postgresql.org/

[5]https://docs.python.org/3/library/subprocess.html

stage that checks for syntax errors and vulnerabilities through static analysis. This check uses regular expressions and, in the specific case of Python, the Bandit[6] tool to identify security risks. The code is then executed against each test case, and its outputs are compared with the correct solution to generate the final verdict, which is returned to the user. This design enables multi-language evaluation without exposing logic in the browser.

4. Results

4.1. Performance

The performance of the original architecture was entirely dependent on the user's device. Figure 2 shows the total processing time for both system versions. The test was conducted in a simulated low-tier mobile device environment (using Chrome DevTools[7]) across three problems with, respectively, 10, 50, and 100 test cases. To simulate high computational demand, a poorly optimized Python solution was used for the benchmarks. This solution artificially simulated a high-load scenario by using benchmark code that included a deliberately inefficient loop of 100,000 iterations.

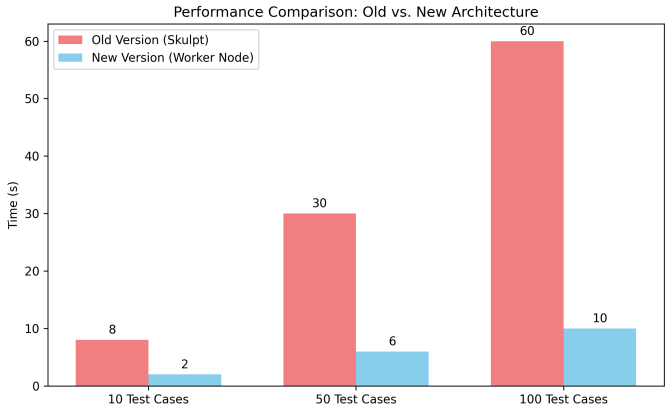

Figure 2. Performance comparison between the old (client-side) and new (server-side) architectures, tested on a simulated low-tier mobile device

The results demonstrate that the new server-side architecture is significantly faster and more efficient, proving it provides more equitable access for users with low-power devices, enhancing accessibility. Furthermore, additional tests simulating poor network conditions showed no significant impact on response times.

4.2. Scalability

To evaluate the new architecture's scalability, load tests were conducted using the Locust[8] framework. The test simulated an extreme scenario of 50 concurrent users continuously submitting Python solutions for 10 minutes, with the system already hosted on Google Cloud Run. The results can be seen in Figure 3.

Even with 50 active users, response times for Python remained at an acceptable level, rarely exceeding 20 seconds. This result confirms the architecture's scalability

[6]https://bandit.readthedocs.io
[7]https://developer.chrome.com/docs/devtools
[8]https://docs.locust.io

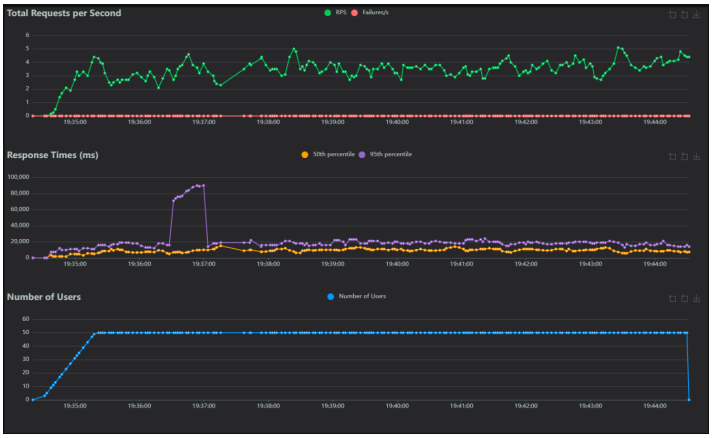

Figure 3. Locust load test results for Python code submissions. The charts display (from top to bottom): requests per second, response times in milliseconds, and the number of active users.

and ability to handle demand spikes, with the C language showing similar performance, whereas Julia's processing did not scale as effectively under the same conditions.

A cost analysis confirmed that, although the 50-user stress test (an atypical and extreme scenario) resulted in a temporary cost spike, the Worker Node's contribution to total project expenses during regular academic periods remained negligible. In these periods, the average monthly infrastructure cost was approximately R$138, with the stateful database VM accounting for about 93.7% and serverless components for roughly 4.5%.

5. Conclusions

The updated MT platform demonstrates how a cloud-native architecture can support more equitable access to educational resources. Future work includes optimizing the Worker Node to reduce processing time for Julia code and integrating AI-driven feedback generation for more didactic and user-friendly responses. We also plan to migrate from Google Cloud to an on-premises distributed infrastructure hosted by partner universities (UFRJ, UNIRIO, and UFRGS) to reduce operating costs and increase operational autonomy.

References

Carvalho, L., Fernandes, D., and Gadelha, B. (2016). Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. In *Anais do XXVII Simpósio Brasileiro de Informática na Educação*, page 140.

Moraes, L., Delgado, C., Freire, J., and Pedreira, C. (2022). Machine teaching: uma ferramenta didática e de análise de dados para suporte a cursos introdutórios de programação. In *Anais do II Simpósio Brasileiro de Educação em Computação*, pages 213–223, Porto Alegre, RS, Brasil. SBC.

Wasik, S., Antczak, M., Badura, J., Laskowski, A., and Sternal, T. (2018). A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)*, 51(1):1–34.

Watanobe, Y., Rahman, M., Matsumoto, T., Rage, U., and Penugonda, R. (2022). Online judge system: Requirements, architecture, and experiences. *International Journal of Software Engineering and Knowledge Engineering*, 32:1–30.