



Practice Paper

Recommended Citation

M. Bovel, & H. Remmal (2024). An Automatic Grading System For Large Classes. Proceedings of the 52nd Annual Conference of SEFI, Lausanne, Switzerland. DOI: 10.5281/zenodo.14256923

This Conference Paper is brought to you for open access by the 52st Annual Conference of the European Society for Engineering Education (SEFI) at EPFL in Lausanne, Switzerland. This work is licensed under a Creative Commons Attribution-NonCommercial-Share Alike 4.0 International License.

An Automatic Grading System for Large Classes

Matt Bovel

EPFL

Lausanne, Switzerland

0009-0005-5132-0279

Hamza Remmal

EPFL

Lausanne, Switzerland

0009-0005-0118-1071

Conference Key Areas: *Digital tools and AI in engineering education, Open and online education for engineers*

Keywords: *Automatic Grading, Infrastructure Management, Engineering Education, Grading at Scale*

ABSTRACT

We present the design, development, and evaluation of an automatic grading system that integrates Moodle with a Kubernetes cluster, aiming to be sufficiently general for use across a variety of courses. Experimentation in two large courses demonstrated the system's capability to support diverse programming assignments, handle substantial concurrent user loads, and enhance the educational experience. Future work will focus on system refinements, exploring additional integrations, and open-sourcing the project to benefit the wider educational community.

1. INTRODUCTION

Automatic testing of programming assignments, recognized for its effectiveness in providing prompt feedback to students and enhancing the scalability of grading processes, is increasingly adopted globally. Within our institution, this trend is reflected across numerous large courses, where we observed similarities in grading processes. We discussed with instructors collectively teaching over fifteen courses across various faculties, each using their own distinct infrastructures, although performing similar functions: students submit their source code for evaluation, feedback is generated on a remote machine and communicated back to them. Developing and maintaining these infrastructures represents a significant investment. Additionally, the presence of multiple systems requires students to adapt to a wide range of user interfaces, workflows, and reporting systems.

This situation highlighted the necessity for a unified system that could be adopted across multiple courses to reduce operational costs and enhance the overall student experience. We identified that an appropriate system would need to be:

- **General:** Not tied to a programming language or assignment format.
- **Simple:** Easy to operate by the students, teachers and system administrators.
- **Automatic:** No human intervention should be necessary for grading.
- **Scalable:** It must be possible to handle thousands of concurrent users, which is crucial to be used during exams and peak submission times.
- **Local:** Students' data must remain within the private network of our institution.

Many existing open-source and commercial solutions (discussed in section 7) do not meet all of these requirements, often being either too specialised, challenging for first-year or non-CS students (e.g., solutions based on Git), or necessitating the outsourcing of data handling.

This inadequacy has prompted us to design and build our own system tailored to these needs, which we present in this practice paper. We evaluated our system over a semester within two significant courses:

- **CS-107: Introduction to Programming (503 students):** The first programming course in the computer science curriculum at our institution. It was important for students to have a simple user interface (UI) since most of them are not yet familiar with tools such as git. Our system was used to automatically grade a self-assessment project and to verify the integrity and proper structure of a larger project.
- **CS-214: Software Construction (383 students):** A second-year course that introduces functional programming, software engineering principles, and formal verification concepts. Here, students completed weekly programming

assignments, which were graded using our system. Additionally, our system has also been used to grade its final exam.

In Section 2, we begin by providing a high-level description of the system, followed by details on its implementation in Section 3. We then describe some applications tested in pilot courses in Section 4. The performance and usability of the system are evaluated in Section 5. Finally, future improvements are outlined in Section 6, and we compare our system with alternative solutions in section 7.

2. SYSTEM OVERVIEW

Our system integrates *Moodle* with a *Kubernetes* cluster and our key contribution is establishing a bridge between the two.

Moodle Plugin: Our system extends Moodle through a custom plugin.

Moodle (Dougiamas and Taylor, 2003) is an open-source Learning Management System (LMS) and the officially supported tool for collecting submissions and distributing online material in many universities. It enables educators to create different content types through *Activities*, including static content as well as dynamic content such as quizzes, polls, and assignments. Comparative studies by Kumar, Gankotiya, and Dutta (2011), along with a systematic review by Gamage, Ayres, and Behrend (2022), have underlined Moodle's efficacy and its evolving role in e-learning. Its widespread adoption among faculty and student body and built-in support for useful settings (e.g., the number of allowed attempts and the submission time window) make it a good foundation to build upon.

Kubernetes Cluster: When students submit their work, our Moodle plugin creates a job within a Kubernetes cluster to run a *Docker image* provided by the teacher. Docker (Merkel, Dirk. 2014) is a virtualization technology that has seen widespread adoption over the past decade. Docker allows for the definition of computing environments, known as images, specified in text files. Kubernetes (Hightower, Burns, and Beda 2017) facilitates the creation of clusters where these images can be efficiently deployed and managed. Basing our interface on Docker images gives teachers complete freedom to run arbitrary programs or workflows for grading student submissions, allowing for a diverse range of assessments possibly beyond programming assignments.

Web Service: The Moodle plugin communicates with the Kubernetes cluster through a small web service. This additional component enables more fine-grained access policies than a direct connection between Moodle and Kubernetes would allow, and also enhances system modularity.

3. IMPLEMENTATION

We extend Moodle's capabilities; and specifically the *Assignments* activity, with a plugin. This plugin allows us to hook into certain events such as submissions by students and assignments creation or deletion by teachers. In particular, the plugin performs an HTTP request to our web service, notifying it of a new submission ❶. In addition to defining webhooks, our plugin extends the Moodle API by providing an additional HTTP endpoints that allows downloading the submission ❺ and uploading grades and feedback to submissions being graded ❹. This endpoint is secured using Moodle's *web services authentication* mechanism.

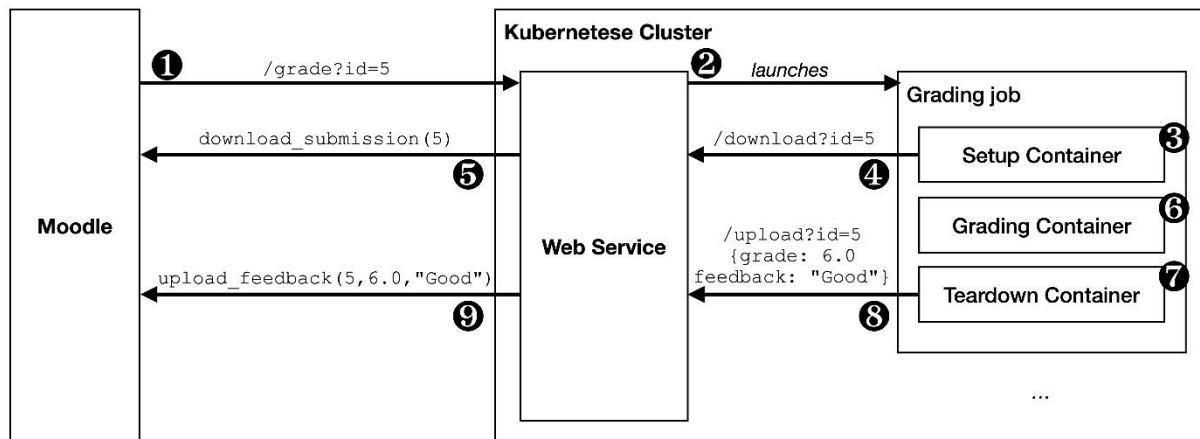


Figure 1: Step-by-step workflow: from the submission to the feedback

As the receiving end of the requests, our service defines a minimal API. On grading requests ❶, the service proceeds by creating a Kubernetes Job ❷; a scheduled task, that will safely download the submission ❹, execute the grading environment ❸ and upload the feedback to Moodle ❾. To enable the possibility to replicate the service, our system avoids storing the state. As such, we currently do not use any database. Authentication is performed by comparing the provided token by a token initialised during the startup of the service. To provide *high-availability* and increase the throughput, the webservice is replicated over multiple nodes of the cluster. On the Kubernetes side, the scheduled job is an aggregate of three *containers* running sequentially. While each container is isolated from the others, they are linked by a shared folder mounted into their file systems, allowing them to share data. The first container ❸ is responsible for downloading the submission ❹ and preparing the grading by putting the files in a dedicated folder in the shared space. The second container ❸; an instance of the provided Docker image, is responsible for running and generating the feedback. It is under the full control of teachers, and it can execute arbitrary code. We require from this executed image to store the feedback in a specific folder of the shared resource. This small contract, with the ability to fully define the grading environment, makes this system *general*. The last container ❹ will upload the feedback and grade to Moodle by calling a specific endpoint of the webservice ❾.

To secure the system, we make sure that all the endpoints of the webservice are protected. As such, we require for each request an authentication token to be stored in the header. Alongside it, and for internally used endpoints ④⑧—which are not meant to be used by users— an additional parameter consisting of a digest of both the context and an internal secret is required. Requests to internal endpoints are generated by the service itself when building the job; specifically in the definition of the setup ③ and teardown ⑦ containers.

4. APPLICATIONS

Our system is agnostic on the format of submissions—files submitted via Moodle are directly forwarded to the grading jobs, which can perform arbitrary operations before generating feedback. This approach opens the room to a variety of applications. In this section, we outline a few of them that were tried during the test phase of our system.

Both CS-107 and CS-214 make extensive use of unit testing. For these cases, we employ a wrapper script on top of the testing framework—JGrade¹ for CS-107 and MUnit for CS-214—to count points and extract feedback. For first-year students in CS-107, we enhance the feedback by displaying test results in an HTML document, which makes navigation easier. In contrast, second-year students in CS-214 are presented with direct text outputs.

Feedback solely based on input-output tests has limitations, especially for solutions that are unusual, whether in a good or in a bad way. For instance, different implementations of the greatest common divisor function for two natural numbers, despite having identical input-output behaviours, might embody distinct algorithmic approaches (e.g., subtraction-based vs. modulo-based Euclid's algorithm). Milovancevic et al. (2024) conducted a trial in the CS-214 course to address this problem, employing our system with a custom image that integrates formal verification of students' code. Specifically, they extended the Stainless verifier (Hamza, Voirol, and Kunčak 2019) to compute equivalences between student submissions and reference solutions. This system not only verifies the correctness of solutions but also analyses and clusters submissions.

In CS-214, students learn the concept of equational reasoning with programs. This primarily consists in transforming simple code expressions using a given set of axioms, emphasising the mathematical foundations of programming. Checking these proofs is also an opportunity for automatic grading. To do so, the teaching staff developed a small surface syntax on top of the LISA interactive theorem prover

¹ JGrade is an in-house library built for grading on top of JUnit 5. Authors are planning to make it open-source in the future.

(Guilloud, Gambhir, and Kunčák 2023), enabling students to write their proofs similarly as they do in traditional paper exercises.

Beyond grading code, another application example is the teaching of *git*. CS-214 teaches students to generate and manipulate *git* patches—text files containing the differences between two project versions. Adapting the grading image for this course to automatically test and apply these patches was straightforward.

5. EVALUATION

Our system allowed course teams to focus on developing the grading logic rather than on infrastructure management. Moodle instance and the Kubernetes cluster were maintained by system administrators, and the maintenance required for the plugin and web service after this initial development phase was low. Early feedback from the teaching teams suggests that routine maintenance tasks, such as updating tokens or adjusting to Moodle updates, should require roughly one to two days of work per semester for a student.

We estimate the development cost of our system at around \$16,000². The running costs of the Kubernetes cluster is not known in our case because it was part of the shared resources of our institution, but we estimate that a similar setup running on Amazon EC2 would cost approximately \$9,000 per semester³.

One of the goals of our system is to efficiently manage the load and to be secure enough to be used during exams. We had the opportunity to validate these aspects during the final exam of the CS-214 course. There, we successfully handled 1715 submissions from 383 students across two batches, including about 600 submissions in the final 15 minutes of each batch with no failures and no delays. A potential drawback of our direct use of Kubernetes jobs worth noting is that it does not use first-in-first-out scheduling, but this was not an issue for us as we provisioned enough resources to avoid queuing delays. If needed, there exists systems that implement this on top of Kubernetes⁴. Additional logistics aspects of organising this exam can be found in (Chiplunkar and Pit-Claudel 2024) .

² The bulk of the expense was attributed to the development work, which was undertaken by students for a total of 450 hours at a rate of \$25 per hour. Additionally, domain specialists installed and configured the Moodle plugin, and created and administered the Kubernetes cluster, amounting to about 20 hours of work at \$80 per hour. Project supervision involved around 80 hours from a PhD student at \$25 per hour and 20 hours from a staff member at \$80 per hour.

³ Our Kubernetes setup comprised 3 admin nodes, each with 4 cores and 8 GB of RAM, and 5 worker nodes, each with 8 cores and 8GB of RAM, for the semester's duration. For the CS-214 exam, we increased the number of worker nodes to 13 for 20 days. Running on Amazon EC2 instances would incur approximately the following costs for 6 months: 3 c6g.xlarge instances for admin nodes at \$1660 and 5 c6g.2xlarge instances at \$5540 for six months, assuming a 1-year reservation paid upfront. For the exam period, an additional 8 c6g.2xlarge instances for one month would cost \$1600. The total cost per semester would therefore be \$8800.

⁴ For example, [kueue](#) is a Kubernetes queuing system.

While we lack comprehensive quantitative data on student usability, informal feedback indicates that a large majority of students found the tool user-friendly. On a more concrete note, an analysis of the CS-214 final exam data revealed that out of 1367 submissions (counting only the latest attempts), 31 had missing files, 10 were left in draft mode, and 18 were complete yet unsubmitted, indicating that approximately 4% of students encountered difficulties. These challenges, especially under the stress of exam conditions, highlight areas for potential improvements in the interface's simplicity that we plan to address (see Section 6.).

6. FUTURE WORK

In this section, we describe a few directions in which our work can be improved.

We are actively documenting the system; from the development, installation by system administrators to the usage by students and teachers, including the development of grading images. We also plan to test the system in more classes and to open-source for other schools to use.

In addition to the informal feedback we already got, the natural next step will be to collect more qualitative data and conduct surveys among the students and the teaching staff, which we plan to do during the second trial phase of our tool.

We are also implementing new features and usability improvements, both for students and teachers. Notably, we plan to add the possibility to check the current status of a submission—for example whenever it is queued or has timed out—from Moodle's interface. Additionally, we plan to integrate more monitoring and management features for teachers, for example allowing them to access raw logs of the containers. To facilitate this, we might switch to a custom Moodle activity in the future, instead of extending the *Assignments* activity.

As it is, the integration with Moodle only allows us to fully grade the project. One use case would be to allow the system to partially grade the submission; by validating the submitted code and delay the grading process for later; e.g. to be graded manually.

We will also investigate the possibility of extending the frontend to support additional LMS. Advanced classes might also benefit by allowing their students to integrate the system with their workflows. For example, by providing students with specific submission tokens and a dedicated API, they would be able write their own submission script or even define custom workflows using *GitHub Actions* or GitLab's equivalent.

7. ALTERNATIVES AND RELATED WORK

Prior to the widespread adoption of Docker and Kubernetes, Danutama and Liem (2013) developed a system using custom sandboxes for isolation. Peveler,

Maicus, and Cutler (2019) analysed Docker-based isolation versus jailed sandboxes for automatic grading, implementing both and concluding that Docker offers stronger isolation guarantees with minimal performance overhead. Calderón, Petersen, and Rodas (2020) introduced SALP, a project aligning with our goals of modularity and generality, albeit as a standalone system rather than integrating with an LMS. Rao (2019) discussed a system similar to ours but tailored for the Canvas LMS. Tafani-Dereeper (2017) designed a system integrating Moodle with RabbitMQ and a set of workers using LTI, which we avoided due to its unsuitability for asynchronous grading and our preference for direct Moodle integration to leverage existing assignment activity features. [A+](#) represents a comprehensive, albeit complex, solution with extensive functionality, including a Moodle plugin last updated in 2021. Unlike these systems, which involve states, databases, separate authentication systems, or UIs, our solution is more minimalistic and has less components to administer and maintain.

For the predecessors of the CS-214 course, we initially adopted GitLab, creating individual repositories for each student and leveraging custom runners for continuous integration. This method, however, presented challenges, particularly during exams, with performance issues leading to outages on both the frontend and Git servers. The recommended GitLab setup for handling 1 push per second with up to 3000 users—a 27-node system—proves both costly and overly complex for our needs. Alternatively, GitHub Classrooms, a system designed to streamline the use of GitHub for educational purposes, was considered. It simplifies repository management and student submissions but lacks the capacity for detailed points allocation. While direct usage of GitHub Actions could potentially address some of these limitations, using it requires additional effort and lacks the user-friendly interface advantages. Tu et al. (2022) highlighted similar constraints.

Commercial platforms like [Ed Lessons](#) and [Gradescope](#) offer web-based interfaces for coding, and manual or automated grading of programming assignments, with Gradescope additionally featuring a GitHub plugin and plagiarism detection capabilities. However, these are separate systems that educators must familiarise themselves with, and they raise concerns regarding sensitive data ownership.

8. CONCLUSION

We described the architecture of an automatic grading system in Sections 2 and 3. By leveraging standard tools such as Moodle and Kubernetes, we developed the system with minimal effort. We evaluated its effectiveness through pilot tests in two courses totalling about a thousand students, including one where it was used for the final exam. The system demonstrated its versatility by supporting several programming languages and assignment types, as evidenced in Section 4. Furthermore, we confirmed its scalability and practical usability in real-world scenarios in Section 5. Looking ahead, we aim to refine our system based on user

feedback, pursue further integrations, and make our solution open-source, as discussed in Section 6.

ACKNOWLEDGMENTS

We would like to thank Dimi Racordon, Barbara Jobstmann, Jamila Sam, Samuel Chassot, and Marwan Azuz for their useful reviews and advice. We are also grateful to the entire teaching teams of CS-107, CS-214, and CS-320 at EPFL, in particular to Clément Pit-Claudel for valuable discussions and guidance, to Sankalp Gambhir for the developing the equational reasoning grader, and to Yichen Xu and Yawen Guan, who organised the CS-214 final exam and provided insightful feedback. Additionally, we appreciate the efforts of everyone else who contributed to developing and refining the tool, including Didier Berguerand, Carlos Perez, Fabien Salvi, Dixit Sabharwal, Benoît Morawiec, Patrick Jermann, Cécile Hardebolle, and Pierre-Olivier Vallès. This project was funded by the DRIL funds from EPFL.

REFERENCES

- Calderón, Diego, Erick Petersen, and Oscar Rodas. 2020. 'SALP: A Scalable Autograder System for Learning Programming - A Work in Progress'. In 2020 IEEE Integrated STEM Education Conference (ISEC), 1–4.
<https://doi.org/10.1109/ISEC49744.2020.9280574>.
- Chiplunkar, Shardul, and Clément Pit-Claudel. 2024. 'Lessons Learned from the First Edition of a Second-Year Course in Software Construction'. European Society for Engineering Education (SEFI).
- Danutama, Karol, and Inggriani Liem. 2013. 'Scalable Autograder and LMS Integration'. Procedia Technology, 4th International Conference on Electrical Engineering and Informatics, ICEEI 2013, 11 (January): 388–95.
<https://doi.org/10.1016/j.protcy.2013.12.207>.
- Dougiamas, Martin, and Peter Taylor. 2003. 'Moodle: Using Learning Communities to Create an Open Source Course Management System'. 171–78. Chesapeake, VA, USA. <https://research.moodle.org/33/>.
- Gamage, Sithara H. P. W., Jennifer R. Ayres, and Monica B. Behrend. 2022. 'A Systematic Review on Trends in Using Moodle for Teaching and Learning'. International Journal of STEM Education 9 (1): 9. <https://doi.org/10.1186/s40594-021-00323-x>.
- Gordon, Chelsea L., Roman Lysecky, and Frank Vahid. 2021. 'The Rise of Program Auto-Grading in Introductory CS Courses: A Case Study of zyLabs'.
<https://peer.asee.org/the-rise-of-program-auto-grading-in-introductory-cs-courses-a-case-study-of-zylabs>.

Guilloud, Simon, Sankalp Gambhir, and Viktor Kunčák. 2023. 'LISA - A Modern Proof System'. In DROPS-IDN/v2/Document/10.4230/LIPIcs.ITP.2023.17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

<https://doi.org/10.4230/LIPIcs.ITP.2023.17>.

Hagerer, Gerhard, Laura Lahesoo, Miriam Anschütz, Stephan Krusche, and Georg Groh. 2021. 'An Analysis of Programming Course Evaluations Before and After the Introduction of an Autograder'. In 2021 19th International Conference on Information Technology Based Higher Education and Training (ITHET), 01–09.

<https://doi.org/10.1109/ITHET50392.2021.9759809>.

Hamza, Jad, Nicolas Voirol, and Viktor Kunčák. 2019. 'System FR: Formalized Foundations for the Stainless Verifier'. Proceedings of the ACM on Programming Languages 3 (OOPSLA): 166:1-166:30. <https://doi.org/10.1145/3360592>

Kumar, Sheo, Anil Kumar Gankotiya, and Kamlesh Dutta. 2011. 'A Comparative Study of Moodle with Other E-Learning Systems'. In 2011 3rd International Conference on Electronics Computer Technology, 5:414–18.

<https://doi.org/10.1109/ICECTECH.2011.5942032>.

Lee, Haden Hooyeon. 2021. 'Effectiveness of Real-Time Feedback and Instructive Hints in Graduate CS Courses via Automated Grading System'. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, 101–7.

SIGCSE '21. New York, NY, USA: Association for Computing Machinery.

<https://doi.org/10.1145/3408877.3432463>.

Milovancevic, Dragana, Mario Bucev, Marcin Wojnarowski, Samuel Chassot, and Viktor Kuncak. 2024. 'Formal Autograding in a Classroom (Experience Report)'. <https://infoscience.epfl.ch/record/309386>

Peveler, Matthew, Evan Maicus, and Barbara Cutler. 2019. 'Comparing Jailed Sandboxes vs Containers Within an Autograding System'. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education, 139–45. SIGCSE '19. New York, NY, USA: Association for Computing Machinery.

<https://doi.org/10.1145/3287324.3287507>

Rao, Dhananjai M. 2019. 'Experiences With Auto-Grading in a Systems Course'. In 2019 IEEE Frontiers in Education Conference (FIE), 1–8.

<https://doi.org/10.1109/FIE43999.2019.9028450>