Improving the Security of Jupyter Autograding

Sami Laine

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 2024-03-08

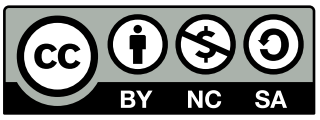
Supervisor

Dr Sanna Suoranta

Advisor

Dr Richard Darst

Aalto University
School of Science



Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Sami Laine

Title Improving the Security of Jupyter Autograding

Degree programme Computer, Communication and Information Sciences

Major Security and Cloud Computing **Code of major** SCI3084

Supervisor Dr Sanna Suoranta

Advisor Dr Richard Darst

Date 2024-03-08 **Number of pages** 38 **Language** English

Abstract

Jupyter Notebooks are a popular tool for science. They are commonly used in education to distribute and grade assignments. When teaching large courses, teachers often use automatic grading because it allows the course staff to focus on tutoring instead of grading. A commonly used tool for automatic grading of Notebooks is Nbgrader, which makes it possible to distribute, collect, and grade the assignments. However, the default grading process of Nbgrader executes the code on the grading teacher's own user account, which introduces significant security risks.

This thesis presents a method to run the grading in a secure, easy-to-use, and fast manner by using a Kubernetes cluster. In this thesis, a proof of concept is created for extending the grading functionality of Nbgrader, to run the grading in a container that is isolated from the teacher's user account. The grading process is run in a container that is created and managed by Kubernetes, which provides a secure and scalable environment for running the grading process. This allows the grading process to be isolated from the teacher's user account and the filesystem of the course platform, which significantly reduces the security risks of the grading process.

This new grading process allows Nbgrader to be used in a more secure manner than it has been previously possible, and with further development, it can be integrated into a Jupyter environment running in a Kubernetes cluster.

Keywords Jupyter, Notebook, Autograding, Automatic grading, Security

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Sami Laine

Työn nimi Jupyter-automaattiarvionnin turvallisuuden parantaminen

Koulutusohjelma Computer, Communication and Information Sciences

Pääaine Security and Cloud Computing **Pääaineen koodi** SCI3084

Työn valvoja TkT Sanna Suoranta

Työn ohjaaja TkT Richard Darst

Päivämäärä 2024-03-08 **Sivumäärä** 38 **Kieli** Englanti

Tiivistelmä

Jupyter-muistikirjat (Jupyter Notebook) ovat suosittu työkalu tieteessä. Niitä käytetään opetuksen apuna tehtävien jakamisessa ja arvioinnissa. Suuria kursseja opettaessaan opettajat käyttävät usein automaattista arviointia, koska se antaa kurssin henkilökunnalle mahdollisuuden keskittyä arvioinnin sijaan opiskelijoiden ohjaukseen. Yleisesti käytetty työkalu Jupyter-muistikirjojen automaattiseen arviointiin onn Nbgrader, jonka avulla tehtäviä voidaan jakaa, kerätä ja arvioida. Nbgraderin oletusarviointiprosessi kuitenkin suorittaa koodin arvioijan käyttäjätilillä, mikä aiheuttaa merkittäviä turvallisuusriskejä.

Tässä diplomityössä esitellään menetelmä, jolla arviointi voidaan suorittaa turvallisella, helppokäyttöisellä ja nopella tavalla, Kubernetes-klusteria hyödyntäen. Diplomityössä luodaan konsepti, joka laajentaa Nbgraderin arviointitoiminnallisuutta, jotta arviointi voidaan suorittaa erillään opettajan käyttäjätilistä, eristetyssä hiekkalaatikossa. Arviointiprosessi suoritetaan Kubernetesin luomassa ja hallinnoimassa kontissa (container), joka tarjoaa turvallisen ja skaalautuvan ympäristön arviointiprosessin suorittamiseen. Tämä mahdollistaa arviointiprosessin eristämisen opettajan käyttäjätilistä ja kurssialustan tiedostojärjestelmästä, mikä vähentää merkittävästi arviointiprosessin turvallisuusriskejä.

Tämän uuden arviointiprosessin ansiosta Nbgraderia voidaan käyttää aiempaa turvallisemmalla tavalla, ja jatkamalla kehitystyötä se voidaan integroida Kubernetes-klusterissa toimivaan Jupyter-ympäristöön.

Avainsanat Jupyter, työkirja, automaattiarvostelu, automaattiarviointi, automaattinen arvostelu, automaattinen arviointi, turvallisuus

Contents

Acronyms

CHP Configurable HTTP Proxy

CLI command line interface

CS-IT Aalto University Department of Computer Science IT

DIND Docker in Docker

LMS Learning Management System

NFS Network File System

OCI Open Container Initiative

TDD Test-Driven Development

UI user interface

1 Introduction

Jupyter Notebooks have become popular tools for scientific computing, machine learning and general programming. They allow combining code, documentation and results into a single document that can be easily shared with and reused by other users. In the Aalto University Department of Computer Science, multiple courses utilise Jupyter Notebooks and the on-premises Jupyter deployment in their teaching as a way to hand out course assignments, as well as to provide interactive course material.

The popularity of Jupyter and Jupyter Notebooks can be explained by their interactive nature which allows for rapid debugging, prototyping, and interactive analysis. Jupyter is commonly used in both education and professional work. Notebooks are an effective format for classroom use when utilised as rich interactive lecture notes and as templates for assignments and projects [16].

A tool called Nbgrader, developed by the Jupyter Development Team, helps automate the distributing, collecting, and grading of the assignment notebooks [21]. The users of Nbgrader can run automatic grading as a batch job after the deadline of an assignment. Students benefit from automatic grading on large courses, as the course staff's time can be used for tutoring and interacting with students, instead of grading a large number of programming tasks.

However, to automatically grade assignments in the default configuration of Nbgrader, the teacher has to run the grading job using their personal account. When using the default configuration, this causes unchecked student code to be run with a teacher's privileges, which introduces significant security risks. Running unchecked student code with a teacher's privileges is a security risk because the assignment submission can contain malicious or otherwise dangerous code that can compromise the teacher's account. This is a significant security risk, as the teacher's account can be used to access other students' submissions, the solutions to the assignments, and the hidden tests. The teacher's account can also be used to modify the grading database, which can be used to obtain a better grade through cheating.

The goal of this thesis is to understand the possibilities of secure autograding and create a proof of concept for running the grading in a secure, easy-to-use, and fast manner. The thesis also surveys possible other security considerations that the current architecture of Jupyter and Nbgrader might contain.

This thesis is structured as follows: Chapter 2 introduces the concept of automatic grading and the security considerations of autograding, the Jupyter ecosystem including JupyterLab and JupyterHub, the Nbgrader tool, as well as the necessary containerisation technologies. Chapter 3 contains an overview of the current architecture of Jupyter and Nbgrader at the Aalto University Department of Computer Science, description of how courses are organised using the Jupyter platform, as well as a description of the existing grading process and its issues. Chapter 4 contains a description of the different methods of improving the security of the grading process that are considered and compared

in this thesis. Chapter 5 describes the set of requirements for choosing a method of improvement, the actual chosen design, and the implementation of the modified grading process. Chapter 6 concludes the thesis and discusses the results of the thesis, as well as presents future work.

2 Background

This chapter describes automatic grading, also known as autograding. It also discusses some security considerations of autograding. Finally, the chapter describes the Jupyter ecosystem, the Jupyter grading tool Nbgrader, and the containerisation technologies that are necessary for the secure autograding methods that are considered in this thesis.

2.1 Autograding and security

The term "autograding" refers to the process of automatically grading course assignments. It can be utilised on courses of any size, but it is especially beneficial on large courses where grading all assignments manually would require a significant amount of work. Automatic grading ensures consistent grading between students and can be used to provide immediate feedback to the students. Providing feedback immediately after the student has submitted the assignment is beneficial for the student, as it allows the student to learn from their mistakes and improve their understanding of the subject [8]. The main benefit of autograding is that it frees up the time of course staff to help students, instead of just grading a large number of assignments. However, autograding also introduces security considerations, as the grading process involves running unchecked student code.

Running unchecked student code requires careful consideration of the security implications. As with any system that runs untrusted code, the system can be compromised if the code is not run securely. The code included in the assignments can be malicious or otherwise dangerous, and depending on the grading environment, running the code can potentially lead to data loss, unauthorised access to the teacher's data, the course data, or other students' submissions, as well as the ability to modify the grading results.

2.2 The Jupyter ecosystem

Project Jupyter is an open-source project originating from the IPython Project [20]. The Jupyter ecosystem consists of multiple components that can be used in different combinations. This includes but is not limited to JupyterLab [19], JupyterHub [18], and Jupyter Notebooks. A grading tool called Nbgrader [21] is also commonly used in educational contexts.

Both the Jupyter Notebook interface, originally known as IPython Notebook, and JupyterLab allow the creation and sharing of documents commonly referred to as notebooks. It is important to note that the word "notebook" can be used to refer to both the file format [37] and the web user interface (UI) [34]. This thesis focuses on an environment where JupyterLab is used as the primary UI, and therefore the terms "notebook" and "Jupyter Notebook" are exclusively used to refer to the file format instead of the web UI. When referring to the

now-deprecated web UI, the term "classic Jupyter Notebook interface" is used instead.

Notebooks can contain live code, Markdown and LaTeX formatting, as well as code output including print output, plots, and visualisations. The default UI for viewing and editing Jupyter Notebooks is a web application that renders '.ipynb' files as HTML and utilises an IPython kernel to execute the code. IPython (short for interactive Python) is a command shell – similar to the default command line interface (CLI) of Python – that focuses on interactive execution.

2.2.1 JupyterLab

JupyterLab is a modern interface that was released in 2019 to replace the classic Jupyter Notebook interface. JupyterLab allows the user to have multiple documents open at the same time in a tabbed interface, supports splits between multiple files or files and terminals windows, and overall provides more of an IDE-like experience compared to the classic Jupyter Notebook interface. Most of the development effort is currently focused on JupyterLab, and the classic Jupyter Notebook interface is considered deprecated.

2.2.2 JupyterHub

JupyterHub is a multi-user service that handles authentication and launching of Jupyter Notebook instances. It can be deployed in numerous ways, of which the two officially supported methods are using a single server or Kubernetes. *The Littlest JupyterHub* is a single-server distribution intended for small deployments serving up to one hundred users [38]. For a larger number of users, JupyterHub can be deployed on a Kubernetes cluster [39]. This provides a more scalable solution than a single server installation but is more complicated to set up. The container orchestration technology Kubernetes is described further in Chapter 2.4. There are multiple ways to organise courses in JupyterHub and the methods relevant to this thesis are described in Chapter 3.2.

2.2.3 The Jupyter kernel communication

Jupyter kernels are separate processes that execute the code contained by the notebook. The kernel communicates with the notebook using the ZeroMQ [4] messaging library. The kernel can be run on the same host as the notebook, inside a container, or on a remote host. The kernel receives code from the notebook, executes it, and sends the results back to the notebook [36]. The notebook then displays the results to the user. The kernel can be interrupted or restarted from the notebook interface, and the notebook can be modified and saved without losing the state of the kernel. This allows for an interactive and iterative workflow.

2.3 Nbgrader

Nbgrader is a tool for creating and grading assignments within Jupyter Notebooks [21]. The process of creating an assignment with Nbgrader consists of the following steps:

1. The teacher creates a source notebook that contains the course material, assignments, assignment solutions, and tests for checking the submitted student code. The source notebook is a completely normal notebook and creating one does not require any special tools or knowledge beyond regular notebook usage. The source notebook contains both the autograding tests and the public student-facing version of the content.
    - The course material can contain readable text in both Markdown and LaTeX formats, as well as code examples to be read or executed by the student.
    - The assignments can consist of automatically graded code cells, manually graded code cells, and manually graded free-form text fields.
    - Both the public and hidden tests are used to check the submitted student code.
2. The source notebook is converted to a version that contains only the course material and assignments. The actual test code is stored separately in a database and replaced with IDs referring to the tests.
3. The converted notebook is released to the students.
4. The students fetch the released assignments, and after modifying the fetched notebook, submit their answers.
5. After the assignment submission deadline, the teacher fetches the submissions and grades them either automatically by using the `autograde` utility to execute the student code and the tests defined in the source notebook or by inspecting the notebooks manually. During the automatic grading process, the test code is written back to the notebook, making sure that the student cannot submit modified tests as part of their submission.

Figure 1 shows a schematic of the Nbgrader workflow. The figure describes the flow of assignments from a teacher to the student and back. Words in black arrows are standard terms of Nbgrader steps. The "student computer" is an optional step for courses requiring extra hardware, such as GPU support for advanced machine learning. Most courses operate completely within JupyterHub.

Nbgrader can be used either as a command line interface via the integrated terminal in Jupyter or through the "Formgrader" extension. Formgrader is a fully-fledged web UI that allows the teacher to create, list, and edit assignments

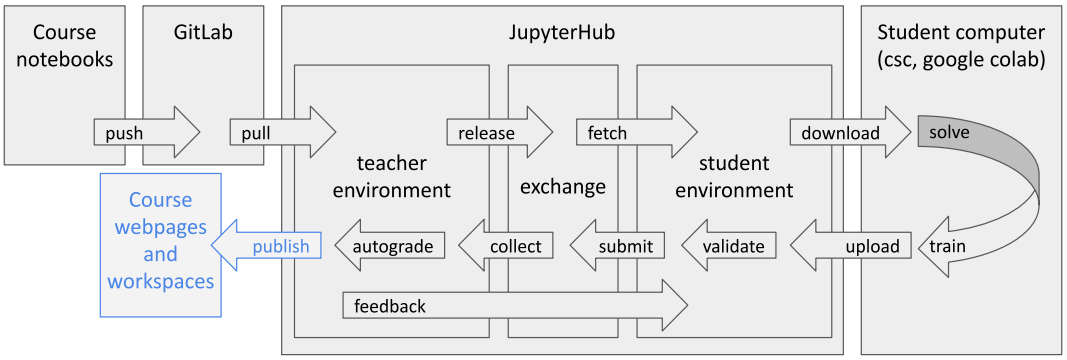


Figure 1: Schematic of Nbgrader workflow describing the flow of assignments from a teacher to the student and back [5].

as well as fetch and release them. It also provides an interface to view and grade student submissions.

When performing autograding with Nbgrader, there is no separate backend service that does the grading. The grading process is done by running all the code in the notebook and looking for errors. This architecture makes it difficult to provide real-time feedback, as the hidden tests get added back to the notebook during autograding. Nbgrader is not designed to support immediate feedback since the grading process exposes the content of the hidden tests as part of the feedback to the students. A common method for implementing automatically graded assignments with Nbgrader, however, is to include both public and hidden tests in the assignment notebook. The public tests are visible to the students and provide a basic level of immediate feedback to the students. The hidden tests are executed after the student has submitted the assignment, and they are ultimately used to check the correctness of the student's code. While it would be possible to create a system that provides real-time feedback, that is not within the scope of this thesis.

2.3.1 Security issues of Nbgrader

Security is one of the minimum standards of any productised service. Within Finland and other countries, the privacy of students is governed by multiple national laws, and platform administrators cannot allow methods for students to access the data of other students, even as a theoretical possibility. In addition, potential unauthorised access to data has negative effects on data integrity.

The default configuration of Nbgrader does not provide a secure way to run the grading process. By default, the notebooks are executed with the same access rights as the user who is grading the notebook. For the grading process, this means running the code on the teachers account. This introduces security risks if the graded code is not manually inspected beforehand. Requiring manual inspection of the code before grading negates the benefits of automatic grading, as the teacher has to spend time inspecting the code before running it.

Running unchecked student code with a teacher's privileges is a security risk

because the code submitted by the student can contain malicious or otherwise dangerous code that can be used to compromise the teacher's account. This is a significant security risk, as the teacher's account can be used to access other students' submissions as well as solutions and hidden tests of future assignments. The teacher's account can also be used to modify the grading database, which can be used to obtain a better grade through cheating. Running unchecked code can also lead to data loss. Unchecked code can potentially be used to access the home directory of the grading teacher as well as the course files.

To avoid the aforementioned flaw in their Jupyter deployment, the Aalto University Computer Science Department IT, from here on referred to as CS-IT, developed a solution for running grading securely in a container on a dedicated server [35]. This solution is based on the `envkernel` [7] library and is described in more detail in Chapter 3.3.

As was described in Chapter 2.2.3, kernels can be executed in different environments. This is often not configured, but it will become a key feature to improving the security. The execution environment used during grading can be configured to be different from the teacher's and the student's normal environments. This is a key feature of the solution that was developed for this thesis.

2.3.2 Grading process failures

Automatically grading Jupyter Notebooks can sometimes be a fragile process. While the JupyterLab interface has some features in place to prevent the notebooks from being modified in unexpected ways, it is still possible to modify the notebook in a way that causes the notebook to fail to execute during the grading process. This can happen, for example, if the student modifies the metadata of a cell within a notebook or deletes the cell that is used to store the student's answer. In these cases, the grading process will fail, and the teacher will have to manually inspect the notebook to determine the cause of the failure.

Common failure points for the grading process include the following:

- Modifying the Nbgrader metadata of a cell within the notebook.
- Deleting the cell that is used to store the student's answer.
- Duplicating cells that contain unique Nbgrader IDs.

Modifying the Nbgrader metadata of a cell within the notebook can cause the grading process to fail. The metadata of a cell is used to store various information about the cell, such as grading ID, as well as the cell type. Listing 1 shows an example of the cell metadata of a student answer cell. Lines 2–5 contain the standard Jupyter cell metadata and lines 6–15 contain the Nbgrader-specific metadata.

Listing 1: Example of a student answer cell metadata.

```
1  {
2    "cell_type": "code",
3    "execution_count": 2,
4    "metadata": {
5      "deletable": false,
6      "nbgrader": {
7        "cell_type": "code",
8        "checksum": "38ef0e9968f48fa9ff18600061c97f71",
9        "grade": false,
10       "grade_id": "square-answer",
11       "locked": false,
12       "schema_version": 3,
13       "solution": true,
14       "task": false
15     }
16     "editable": true,
17   },
18 }
```

The most important fields related to grading failures are the `deletable` field on line 5, the `grade_id` field on line 10, the `locked` field on line 11 and the `editable` field on line 16. Their meanings are explained next.

The `deletable` field is used to indicate whether the cell can be deleted. By default, the JupyterLab interface does not allow the user to delete cells marked with this flag, but with the correct knowledge, it is possible to modify the metadata of a cell to bypass the removal protection. However, during normal use, there is no reason for the student to delete the graded cells deliberately. A more likely situation is that the student modifies the notebook in a Jupyter environment that does not respect the `deletable` flag, such as on a local machine or in a JupyterLab instance different from the course environment, such as Google Colab. In those environments, the protection against deleting special cells used by Nbgrader might not be present, and it is possible to accidentally delete important cells. If the marked cells are not present in the submission, Nbgrader fails verify the content of the cells and the grading process will fail. It is possible for the teacher to edit the submission and re-add the missing cells, but this is a manual process that requires additional work.

The `grade_id` field is used to store the unique ID of the cell. The ID of the cell is used to cross-reference the cell contents with the grading database. If this ID is modified, Nbgrader will fail to detect the cell correctly and the grading process will fail. Additionally, the JupyterLab interface does not prevent the user from copying and pasting cells, even if they would be marked as read-only by Nbgrader. This can cause the grading process to fail, as the submission will contain multiple cells with the same ID.

The `locked` field is used to indicate whether the cell can be edited or not. In

most implementations, setting the `locked` field to `true` also sets the `editable` field to `false`. This flag is usually set for cells that contain automatic tests, in order to prevent the student from modifying the test code. The contents of the test cells are replaced during the grading process with the actual test code that was stored in the grading database. Therefore, locking the cell is only done to prevent the student from accidentally modifying the test code and getting confused by the incorrect test results before submitting the assignment. By default, the JupyterLab interface does not allow the user to edit locked cells when the Nbgrader extension is enabled in the environment, but with enough knowledge, it is possible to modify the metadata of a cell to bypass the edit protection. However, during normal use, there is no reason for the student to edit the locked cells deliberately. A more likely situation is that the student modifies the notebook in a Jupyter environment where Nbgrader is not present, such as on a local machine or in a JupyterLab instance different from the course environment, such as Google Colab.

2.4 Containerisation technologies

Setting up a Jupyter deployment does not inherently require the use of containers, but containers are often used to provide a consistent environment for the students and teachers. A container image designed to run Jupyter includes all the necessary software and dependencies to run the Jupyter Notebook server. This simplifies the deployment process, as the user does not need to install any software on their local machine to access Jupyter, and all teachers and students use the same versions of the software. It is also possible to set up a Jupyter deployment in a Kubernetes cluster for added scalability and reliability. The Jupyter deployment at the Aalto University Department of Computer Science is set up in a Kubernetes cluster, as described in Chapter 3.1. The grading is performed in an environment that is based on Apptainer, a container runtime that allows running containers without root access. The grading environment is described in more detail in Chapter 3.3. The following isolation and containerisation principles and technologies as well as other technologies are discussed in this chapter:

- Sandboxing
- Containers
- Kubernetes and Minikube
- Apptainer, formerly known as Singularity
- Gateway Provisioners

2.4.1 Sandboxing

Sandboxing is a commonly used security mechanism for separating different running processes in a system. It is usually used to mitigate software vulnerabilities [13]. It is often used to execute untrusted code from unverified third parties or untrusted users. The Linux kernel provides a feature called cgroups that allows the creation of sandboxes [15]. Container runtimes use this feature to create isolated environments for running containers.

2.4.2 Containers

Containers are a standardised way of packaging up software so that the code and all its dependencies are included in a single bundle [33]. A container allows an application to be run easily and reliably in different computing environments with very little computational overhead. A container image is a standalone, executable package that contains the necessary information to run a piece of software, including the code, libraries, environment variables, and configuration files.

A container runtime is a piece of software that is responsible for running containers. The Open Container Initiative (OCI) is a standard that defines how containers should be packaged and run [1]. The OCI standard is supported by multiple container runtimes, which allows the containers to be run in different environments without modification. A widely used container runtime is Docker [27], but there are many other container runtimes that are compatible with the OCI standard.

2.4.3 Kubernetes

Kubernetes is an open-source container orchestration platform originally developed by Google for managing containerised workloads and services [30]. It allows running containers in a cluster of multiple worker nodes and provides mechanisms for deploying, maintaining, and scaling services within the cluster [31].

Minikube is a tool that lets the user set up a Kubernetes cluster on a single machine [28]. It is often used for prototyping and development when a full Kubernetes cluster is not necessary.

2.4.4 Apptainer

Apptainer (formerly known as Singularity) is a container runtime that allows running containers without root privileges [3]. It was originally designed for scientific computing tasks and is often used in high-performance computing environments.

2.4.5 Gateway Provisioners

Gateway Provisioners is a library that allows remote execution of Jupyter kernels [10]. It is capable of running Jupyter kernels inside Docker containers, Kubernetes pods, in a Hadoop Yarn cluster, or on a remote server over an SSH connection.

3 Jupyter in Aalto University

This chapter describes the current state of the Jupyter deployment at the Aalto University Department of Computer Science, as well as the current autograding setup. The chapter also describes the key issues with the autograding setup, which will eventually be solved by the methods described in Chapters 4 and 5.

3.1 Technology stack

In Aalto University, a JupyterHub deployment has been set up by utilising a local self-hosted Kubernetes cluster [2]. CS-IT manages the cluster and the deployment.

The deployment consists of JupyterHub and JupyterLab container images that are built on top of the official Jupyter Docker Stacks images. The deployment has been set up by creating manual Kubernetes resource definitions to create a flexible and customised system, fully integrating JupyterHub and its data storage into Aalto University Network File System (NFS) shares. User home directories and public course data are served from a shared file system over NFS. User authentication is performed over OAuth2 via Microsoft Entra ID (previously AzureAD).

Spawn profiles are used to create a course selector, where students can choose from all the different courses that use the platform. Each course can have its customised assignments and course data, all served over NFS to allow a multi-node setup. Some courses use different container images that allow the courses to utilise different packages or versions of the packages.

The grading improvements discussed in this thesis are designed to be independent of the method of deploying the Jupyter environment and therefore could be applied to most JupyterHub deployments running inside a Kubernetes cluster.

3.2 Organising courses in JupyterHub

When running JupyterHub in a Kubernetes cluster, the `KubeSpawner` configuration can be customised to define multiple different spawn profiles which allows the spawning of several distinct types of single-user Jupyter instances [23]. For example, profiles can be used to spawn notebook instances with different container images, resource limits, or Kubernetes node selectors. These profiles can be used to customise the spawned instances for different needs of courses, but this method alone is not suitable for creating isolated environments for autograding.

3.3 Current autograding setup

CS-IT offers a separate grading environment that prevents the issue of teachers running unchecked student code. However, the environment is not directly

accessible to the teachers and utilising it requires manual work and communication between the teachers and the IT staff. In the existing secure grading environment, an administrator creates a dedicated Apptainer container config for each course instance. The container is then used to run the Jupyter kernels for the student-submitted notebooks in a sandboxed environment. Inside the container, the kernel being executed can access the shared course files and the files submitted by each student one at a time, but the code executed by the kernel cannot access or modify any home directories or other course data.

CS-IT has set up a dedicated virtual machine to run the grading process inside a Apptainer container. CS-IT has developed an open-source library Envkernel [7] that allows the running of Jupyter kernels in multiple separate ways, such as inside a Conda environment or a Python virtual environment, in a Docker container or a Apptainer container, or with Lmod [24] modules. When using the virtual machine, the course image is converted from an OCI image to the Apptainer format and added to Jupyter as a separate kernel using the Envkernel library.

When starting the grading process with Nbgrader, the course data is mounted to the Apptainer container and the kernel is run inside the container. No user data is present in the container, which prevents the code from the submitted notebooks from accessing the home directories of the teachers. The current setup is based on and partially documented in the Nbgrader documentation.[1]

3.4 Issues with the autograding setup

The autograding setup described in Chapter 3.3 solves some of the issues with the default Nbgrader setup, but it still has its own shortcomings. With the current setup, teachers cannot trigger the grading process themselves by pressing a button in the teacher's view. Instead, the teachers have to contact the CS-IT and request them to run the grading separately. This introduces unnecessary delays and complications to the grading process, as well as makes the grading dependent on the availability of the IT staff.

[1] https://nbgrader.readthedocs.io/en/stable/user_guide/advanced.html#grading-in-a-docker-container

4 Improving the grading process

The goal of this thesis is to understand the possibilities of secure autograding and to create a proof of concept for running the grading of Jupyter Notebooks in a secure, easy-to-use, and fast manner. First, several broad types of solutions were considered and compared, even some that go beyond the standard Nbgrader paradigm:

- Modifying the current automatic grading process of Nbgrader to allow teachers to run the grading in a separate, isolated environment. This includes either utilising the existing Apptainer-based autograding system that is used in the Aalto University Department of Computer Science, or creating a new system that is based on the same idea.
- Utilising some of the existing third-party tools, such as Grader-Service [12], UNCode [11], or Web-CAT [25] for handling and grading Jupyter Notebooks.
- Utilising the existing Aalto University Learning Management System (LMS) A+ [22] and its automated assessment tool MOOC-Grader [26] for grading Jupyter Notebooks.

In this chapter, we will broadly compare these options and consider in which cases each is most appropriate. Table 1 summarises the main advantages and disadvantages of each method, as well as the subsequent chapters where each method is discussed further. The chosen solution, modifying Nbgrader, is described more in detail in Chapter 5.

4.1 Modifying Nbgrader

We consider multiple different methods that would allow teachers to run the grading in a separate, isolated environment. Modifying Nbgrader is closest to what is already in use, as described in Chapter 3, so there are varying options to incrementally improve the setup. The following options are available for modifying Nbgrader:

- Granting teachers access to the virtual machine that has been used to run Nbgrader by the IT staff
- Running the grading in a separate container inside the teacher's single-user instance
- Spawning dedicated grading pods in the Kubernetes cluster.

The last point is a much more fundamental change than the previous options. Next, we will discuss the advantages and disadvantages of each of these methods.

Table 1: Summary of advantages and disadvantages of different grading methods

Grading method	Chapter	Advantages	Disadvantages
Grading VM	4.1.1	• Most infrastructure exists already • Relatively easy to adapt to a multi-user setup	• Requires teachers to learn a completely new system • Accessed via SSH, the UI is not integrated with Jupyter
Docker-in-Docker	4.1.2	• Accessible from the Jupyter web UI • Can reuse the container image that is used to run the notebook	• Not recommended by the Docker developers • Grading container lifetime is dependent of the teacher's session • Limited to the resources available to the single-user instance • Requires more effort from the administrators' side to set up than the existing grading VM
Kubernetes	4.1.3	• Accessible from the Jupyter web UI • Can utilise the full resources of the cluster • Not tied to the lifetime of the teacher's session • Can reuse the container image that is used to run the notebook • Can parallelise the grading	• Requires more effort from the administrators' side to set up than the existing grading VM
GraderService	4.2.1	• An existing tool • Compatible with the Nbgrader format	• Still in an early development phase • Drastically changes the grading interface

Table 1: Summary of advantages and disadvantages of different grading methods (continued)

Grading method	Chapter	Advantages	Disadvantages
UNCode	4.2.2	• An existing tool • Provides instant feedback	• Does not integrate with JupyterLab • Does not use the Nbgrader format for assignments • Uses a completely separate web UI
Web-CAT	4.2.3	• An existing tool with long development history • Provides instant feedback	• Does not integrate with JupyterLab • Uses a completely separate web UI
MOOC-Grader	4.3	• Could be made accessible from the Jupyter web UI • Can utilise the full resources of the cluster • Not tied to the lifetime of the teacher's session • Can parallelise the grading • Existing and well-maintained tool, no need to implement from scratch	• Highly customised for the grading protocol of A+, almost impossible to adapt

4.1.1 Access to the grading virtual machine

At the start of this project, the actively used method for securely performing the grading is to run the grading process inside a virtual machine that has been set up by CS-IT. This process is described in more detail in Chapter 3.3.

It is possible to grant the teachers an SSH access to the virtual machine and allow them to start the grading process themselves. This is the simplest solution to implement because it only requires minimal changes to the existing setup. The main advantage of this method is that most of the infrastructure already exists and is relatively easy to adapt to a multi-user setup. However, utilising the virtual machine is not particularly user-friendly and would require the teachers to learn a new system that is completely separate from the rest of the Jupyter ecosystem. This could potentially decrease the number of courses that are interested in using automatic grading because of the added complexity.

Additionally, due to the way the virtual machine has previously been set up exclusively for admin use, it allows access to data of all courses and user home directories by default. To prevent this, a stricter access control policy would have to be implemented to ensure that the teachers cannot access other Jupyter users' home directories and that they can only access the course data and submissions for courses they teach.

4.1.2 Running container in a container

The second considered way of isolating the grading jobs from the rest of the system is to run the grading in a separate container inside the single-user instance container, a technology sometimes known as Container-in-Container or Docker in Docker (DIND). This efficiently separates the filesystem of the graded notebook from the teacher's home directory because it is possible to choose which directories, such as the public course data and a single student's submission, are mounted to the grading container.

However, when using this method, the grading job is tied to the lifetime of the single-user instance, which makes it difficult to run the grading in the background. This means that the grading process can be interrupted if the teacher's single-user session terminates, either due to inactivity or due to the teacher switching to a different course instance. The grading process is also limited to the resources available to the single-user instance, although this is not a major issue as the grading process is usually not run in parallel. Moreover, the students have been successfully running the notebook using similar resources before submitting the assignment, so it is likely that the grading job will not be affected by the resource limits.

Ultimately, the main reason for not selecting Docker in Docker as the implementation method is that the original author of the feature, Jérôme Petazzoni, recommends against using it [29]. Petazzoni describes issues with Linux security modules, problems related to storage drivers as well as potential data corruption when using the feature. Petazzoni also suggests that the feature is not necessary for most use cases and that it is better to run containers side by side, instead of nesting them. The Kubernetes equivalent of this recommended method of running containers side by side is described in Chapter 4.1.3.

4.1.3 Grading pods in Kubernetes

The third considered option is to spawn dedicated grading pods in the Kubernetes cluster. This allows the teachers to perform the grading in a completely separate container that is not tied to the teacher's session or filesystem in any way. The teachers can run the grading process in an independent environment and continue using the platform while the grading is running. The grading process can utilise the full resources of the cluster, which could potentially make it possible to parallelise the grading in the future. However, in its current state Nbgrader does not support parallel grading [9]. Similarly to nesting the grading container inside another container as considered in Chapter 4.1.2, the

grading pods can be configured to only mount the necessary directories, which prevents the submitted code from accessing the home directory of the teacher or sensitive course data. This is the solution that was ultimately chosen for further development. The implementation of this solution is described in more detail in Chapter 5.

4.2 Third-party grading tools

Several third-party grading tools have been developed by other universities and organisations that could potentially be used to grade Jupyter Notebooks. These tools are designed to be used in combination with Jupyter Notebooks and could potentially be adapted to work with Nbgrader. The following tools were considered:

- Grader-Service [12]
- UNCode [11]
- Web-CAT [25]

4.2.1 Grader-Service

An Austrian university TU-Wien has developed an open-source tool called Grader-Service that integrates with JupyterLab and utilises Nbgrader to convert Jupyter Notebooks into an automatically gradable format [12]. Grader-Service is designed to be used in combination with the JupyterLab interface, and it allows the teachers to trigger the grading process directly from the JupyterLab interface.

Grader-Serviec is an existing tool that would not require a large amount of development work to integrate with the existing Jupyter environment. It also has the advantage of being compatible with the Nbgrader format, which means that it could be used to grade the existing assignments without requiring the teachers to modify the assignments to fit a new format. However, Grader-Service is still in an early development phase and is not yet widely used, which makes it difficult to evaluate its suitability for the current environment. It also drastically changes the grading interface, which could make it difficult to adapt to the existing workflow of teachers.

4.2.2 UNCode

UNCode is an open-source tool, developed in the Universidad Nacional de Colombia, which is designed to be used for grading programming assignments [11]. UNCode is a web-based tool that allows teachers to create assignments and automatically grade the student submissions. It was originally designed for grading Jupyter Notebooks containing machine learning assignments, but it can be used to grade any type of programming assignment.

UNCode does not utilise JupyterHub or JupyterLab, which makes it a less suitable candidate for grading Jupyter Notebooks in the existing environment. Instead, it includes a standalone web UI that allows teachers to create assignments and students to submit their solutions.

4.2.3 Web-CAT

Web-CAT is an open-source tool for automated testing [32]. It was originally developed in 2003 at the Virginia Polytechnic Institute and State University to aid with Test-Driven Development (TDD) in programming courses. It has since been expanded with contributions from other users. Web-CAT is a web-based tool that includes a standalone web UI. It supports submitting and automatically grading programming assignments, including providing immediate feedback to the students. In 2020, Web-CAT was expanded to support Jupyter Notebooks [25], which makes it a potential candidate for this project.

The Jupyter Notebook support in Web-CAT maintains compatibility with the Nbgrader assignment format, even though it does not use Nbgrader as the grading backend. However, Web-CAT is not designed to be integrated with JupyterHub or JupyterLab, which makes it a less suitable candidate for grading Jupyter Notebooks in the existing environment.

4.3 MOOC-Grader

Aalto University uses an internally developed LMS called A+ [22] and an accompanying automated assessment tool MOOC-Grader [26] for many of its courses, most of which contain programming assignments. MOOC-Grader provides an environment where student submissions can be automatically run and graded in isolated containers. These containers could be modified to run and grade Jupyter Notebooks while utilising the functionality and computational power of the existing MOOC-Grader environment.

The approach of running grading pods in Kubernetes is conceptually somewhat similar to the MOOC-Grader setup used by A+. MOOC-Grader has been used as an inspiration for the resulting design. However, when researching options for implementing the grading process, it became apparent that MOOC-Grader is not a good fit for the Jupyter environment directly. While it would be possible to adapt MOOC-Grader to execute Jupyter notebooks, it relies heavily on the A+ LMS and produces the grading results in an HTML format that is not directly compatible with the design of Nbgrader. MOOC-Grader is designed to suit the needs of and to work very closely with the A+ LMS, which makes it impractical to adapt to Nbgrader.

4.4 Chosen technologies

The methods described in Table 1 were surveyed. Third-party tools such as Web-CAT [25], Grader-Service [12], and UNCode [11] were found to be not

suitable because they did not integrate well with the existing Jupyter and Nbgrader setup. The idea of the existing container-based autograding system was found to be a promising suitable option, but not easily usable in the teacher's environments. An existing open-source library, Gateway Provisioners [10] which had not been previously used in the context of Nbgrader, was chosen for the implementation. The Gateway Provisioners library is further described in Chapter 2.4.5. By combining the idea of containers and Gateway Provisioners, we were able to extend the grading-in-container system so that it could run in the Aalto University JupyterHub production environment by utilising the existing Kubernetes cluster.

The chosen method of implementing the grading process is to spawn dedicated grading pods in the Kubernetes cluster. This allows the teachers to start the grading in the background and continue using the platform while the grading is running. The grading pod is separated from the teacher's user session, which allows the pod to utilise custom resource limits, and it isolates the grading job from the teacher's home directory and other course data. This implementation is further described in Chapter 5.2.

5 Design and Implementation

This thesis performs the groundwork for creating a secure grading environment that is easy to use for both teachers and students. A sophisticated UI is left as future work, and the focus of this thesis is on a functional grading backend.

5.1 Requirements

The requirements that the grading process must satisfy have been set as follows:

- The grading process must be secure.
- The teacher must be able to perform the grading as self-service easily, without requiring help from the department IT.
- The grading process should be reliable.

The requirements of success for this thesis are that the grading system is isolated from the teacher's Jupyter session and filesystem and that the teacher can perform the grading as self-service, without requiring help from CS-IT. There are also secondary requirements that the grading system should eventually satisfy, but these are not strictly necessary for the prototype to be considered successful. The secondary requirements are that the grading process should be reliable and that it should be easy for the teacher to perform the grading. The different requirements are further discussed later in this chapter.

5.1.1 Secure grading process

The first main requirement for resulting grading process is that it must be secure. A secure grading process ensures that it cannot be used to gain unauthorised access to the grading environment or the rest of the Jupyter deployment, as well as provides against accidental loss of data. To ensure security, the grading process must be isolated from the teacher's Jupyter session and filesystem. The teacher must be able to run the grading process without the risk of the submitted code accessing the teacher's home directory or other sensitive course data, such as the grade database.

5.1.2 Self-service for teachers

The second requirement is that the teachers must be able to perform the grading as self-service, without having to manually contact CS-IT after every deadline. Being able to perform the grading as self-service minimises unnecessary delays in the grading process. The teachers can start the automatic grading immediately after the deadline has passed and they can follow the progress of the grading and any arising issues in real time, without CS-IT acting as an intermediary.

Additionally, it should be easy for the teacher to perform the grading. The main requirement for this is that the teacher should not be required to learn a

completely new system to perform the grading. To make the grading process easy to access, it should be integrated into the existing Jupyter deployment in a way that allows the teacher to start the grading process directly from the Jupyter UI.

5.1.3 Reliability of the grading

Finally, the grading process should be reliable. A reliable grading process guarantees that the grading environment is available at all times. Grading should be reliable in a way that it is always available to the teachers, and that the grading process is completed correctly, and possible errors are reported to the teacher. Because of the nature of the current Nbgrader implementation, individual notebooks can fail to grade, but the grading process should still be completed as a whole. If the submitted notebook contains malformed metadata or has other issues, the grading of that notebook should be skipped, and the teacher should be notified.

5.2 Implementing the grading process

In the proof of concept developed in this thesis, the grading process is implemented utilising a separate library called Gateway Provisioners that allows remote execution of Jupyter kernels. The library is described in further detail in Chapter 2.4.5. In principle, this method is similar to the method of using envkernel and Apptainer as described in Chapter 3.3, but to the best of our knowledge, the Gateway Provisioners library has not been used in the context of Nbgrader and autograding before.

The library is used to spawn separate grading pods in the Kubernetes cluster, which are then used to run the grading kernel. The main purpose of the library is to allow the execution of Jupyter kernels in a remote environment for distributing resource-intensive tasks to multiple calculation nodes. However, because of the isolated nature of Kubernetes pods, the library is also suitable for isolating the grading process securely. The grading pods are configured to mount the necessary directories, including the public course data.

As part of the development process, a proof-of-concept solution was deployed and tested in a local Kubernetes cluster. The cluster was set up on a single node using Minikube. The solution was tested with an example course that contained an assignment with several test cases.

5.2.1 Setting up the grading

The default method of installing the Gateway Provisioners library is to use the Python package manager PIP [6] with the command `pip install gateway-provisioners`. After the library has been installed, Jupyter is configured to use the Kubernetes cluster as the execution environment for the kernels by running the command `jupyter k8s-spec install`.

Listing 2 shows the Dockerfile that is used to create the JupyterLab container image. The image is based on the `jupyter-aalto-singleuser` [14] image, which in turn is based on the `minimal-notebook` version of the official Jupyter Docker Stacks images [17]. The official Jupyter images contain the basic tools such as pre-installed Conda, Python, and the Jupyter libraries and their dependencies, which makes them a good starting point for creating custom Jupyter images. The image is configured to use the `k8s_python` kernel provided by the Gateway Provisioners library to run the notebook code in the Kubernetes cluster.

Listing 2: A Dockerfile for creating the JupyterLab container image.

```
1  # Base image built from
2  # https://github.com/AaltoSciComp/jupyter-aalto-
       singleuser/tree/v6.3
3  FROM docker.io/aaltoscienceit/notebook-server-base
       :6.3
4
5  USER root
6  CMD ["/bin/bash"]
7  EXPOSE 8888
8  WORKDIR /home/jupyter
9  ADD start-jupyterlab.sh /opt/start-jupyterlab.sh
10 RUN chown $NB_UID:$NB_GID /opt/start-jupyterlab.sh
11 USER $NB_UID
12 ENTRYPOINT sh /opt/start-jupyterlab.sh
13
14 RUN pip install "gateway_provisioners[k8s]"
15
16 # Removed because causes issues with the version of
17 # jupyter_server that is installed in the
18 # notebook-server-base image
19 RUN pip uninstall jupyter_server_terminals -y
20
21 # --sys-prefix: Install globally in the conda
22 # environment
23 RUN jupyter k8s-spec install --sys-prefix
24
25 # A script included in notebook-server-base that
       enables
26 # the Formgrader and nbgrader extensions
27 RUN enable_formgrader.sh
28
29 # Use the full path to the ‘python‘ executable,
       because
30 # ‘nbgrader autograde‘ doesn’t seem to include
```

```
31 # /opt/conda/bin in PATH
32 RUN sed -i 's;  "python",;  "/opt/conda/bin/python
      ",;' \
33     /opt/conda/share/jupyter/kernels/k8s_python/
         kernel.json
34
35 # Add /coursedata volume mount to the kernel
36 RUN sed -i 's;"env": {},;"env": {\n    "
      KERNEL_VOLUME_MOUNTS": "[{\\"name\\": \\"coursedata
      \\", \\"mountPath\\": \\"/coursedata\\", \\"
      readOnly\\": true}]",\n    "KERNEL_VOLUMES": "[{\\"
      name\\": \\"coursedata\\", \\"hostPath\\": {\\"path
      \\": \\"/data/coursedata\\", \\"type\\": \\"
      DirectoryOrCreate\\"}}]"\n  },;' \
37     /opt/conda/share/jupyter/kernels/k8s_python/
         kernel.json
```

Listing 3: The `start-jupyterlab.sh` file for starting the JupyterLab server in a container.

```
1 #!/bin/bash
2
3 jupyter lab --no-browser \
4              --allow-root \
5              --ip=0.0.0.0 \
6              --NotebookApp.token='' \
7              --NotebookApp.password=''
8
9 tail -f /dev/null
```

Listing 3 shows the `start-jupyterlab.sh` file that is added to the container image in Listing 2 line 12. The script file `start-jupyterlab.sh` is used as an entry point to the container and it starts the JupyterLab server.

The Gateway Provisioners library is used to spawn separate grading pods in the Kubernetes cluster, which are then used to run the grading kernel. The library is configured as a separate kernel in the `kernel.json` file and running the command on line 23 of Listing 2.

The pod running the kernel is configured to mount the necessary directory: the public course data. The environment variables that are used to configure the kernel are set on line 36 of Listing 2 by replacing the empty environment variable dictionary `"env": {},` with the desired variables in the `kernel.json` file. Listing 4 shows a formatted version of the environment variables dictionary that is placed in the `kernel.json` file. Listings 5 and 6 further show the formatted versions of the content of the environment variables `KERNEL_VOLUME_MOUNTS` and `KERNEL_VOLUMES` that are used to set up the directory mount. These correspond to the Kubernetes container definition directives `volumeMounts` and

`volumes`, respectively. In the test environment, the data is mounted from a local directory on the Kubernetes node, whereas in the production environment, the data will be mounted from a NFS server.

Listing 4: Setting up mount environment variables for the kernel. Decoded from line 36 of Listing 2.

```
1 {
2   [..]
3   "env": {
4     "KERNEL_VOLUME_MOUNTS": "[{\"name\": \"coursedata
        \", \"mountPath\": \"/coursedata\", \"readOnly
        \": true}]",
5     "KERNEL_VOLUMES": "[{\"name\": \"coursedata\", \"
        hostPath\": {\"path\": \"/data/coursedata\", \"
        type\": \"DirectoryOrCreate\"}}]
6   },
7   [..]
8 }
```

Listing 5: Decoded `KERNEL_VOLUME_MOUNTS` from line 4 of Listing 4.

```
1 [
2   {
3     "name": "coursedata",
4     "mountPath": "/coursedata",
5     "readOnly": true
6   }
7 ]
```

Listing 6: Decoded `KERNEL_VOLUMES` from line 5 of Listing 4.

```
1 [
2   {
3     "name": "coursedata",
4     "hostPath": {
5       "path": "/data/coursedata",
6       "type": "DirectoryOrCreate"
7     }
8   }
9 ]
```

5.2.2 Jupyter architecture

Figure 2 shows a schematic of the interactions between JupyterHub, JupyterLab, and the Jupyter kernels in the Jupyter deployment at Aalto University. The labelled rectangles represent Kubernetes pods and the arrows represent the flow

of data and control. The rectangle with rounded corners called Configurable HTTP Proxy (CHP) represents a proxy process launched by JupyterHub that forwards network traffic between the web browser and JupyterHub and JupyterLab. In the arrows, the "(rw)" label indicates that the directory is mounted as read-write, and the "(ro)" label indicates that the directory is mounted as read-only. In the test environment, the NFS server is replaced with a local directory on the Kubernetes node.

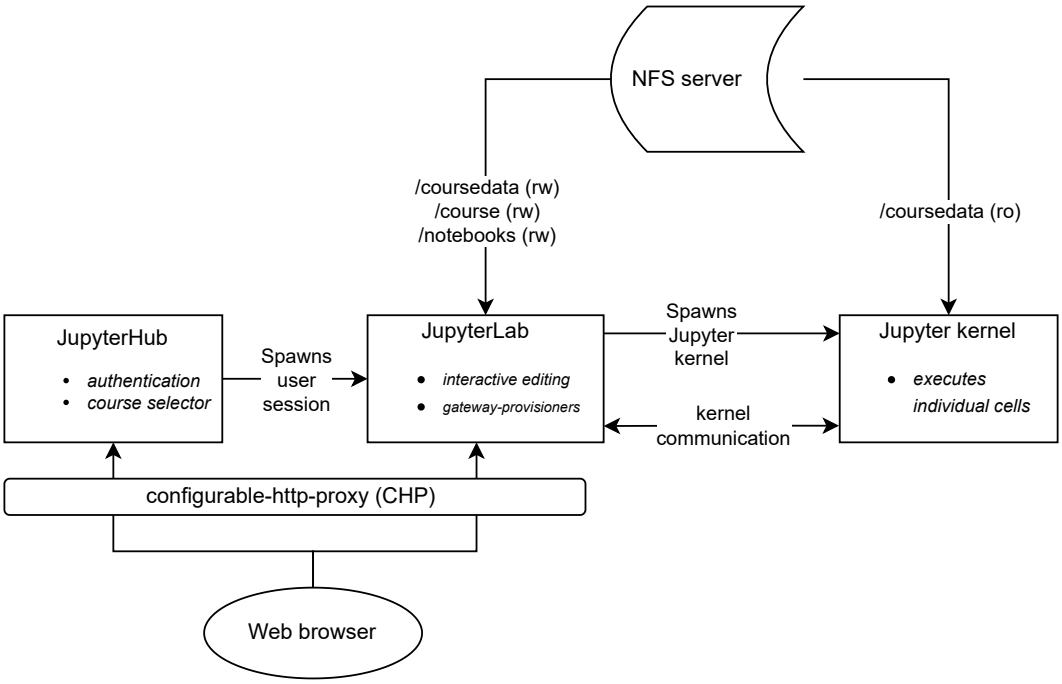


Figure 2: Schematic of interactions between JupyterHub, JupyterLab, and the Jupyter kernels.

The JupyterHub deployment is responsible for authenticating the users and spawning the single-user instances. A teacher logs in to the JupyterHub interface using their university credentials and chooses the course instance they want to access. JupyterHub spawns a single-user instance for the user. The start-up command of the single-user pod launches the JupyterLab interface which is used by the user to interact with the Jupyter environment.

Via the JupyterLab interface, the teacher can start the grading process by using Nbgrader and can inspect the results of the grading. When the grading process is started, Nbgrader utilises the Gateway Provisioners library to spawn a grading pod in the Kubernetes cluster. The grading pod runs the Jupyter kernel, which is responsible for executing the code contained by the individual cells in the notebook that is being graded.

5.3 Results

The requirements of success for this thesis were introduced in chapter 5.1. They are further evaluated in this chapter.

5.3.1 Secure grading process

The teacher must be able to run the grading process without the risk of the submitted code accessing the teacher's home directory or other sensitive course data, such as the grade database. This isolation is achieved by running the grading process in a separate pod in the Kubernetes cluster. This way the graded code can only access files and directories that have been explicitly mounted to the grading pod. These files include the public course data and the files extra copied as part of the student's submission. The extra files can include datasets and other files that are required for the notebook to run but are not part of the notebook itself.

5.3.2 Grading as self-service and easy grading

The teachers must be able to perform the grading as self-service, without the help of the department IT. This is achieved by integrating the grading process into the existing Jupyter interface. The teachers can start the grading process for individual assignments by clicking a button in the Formgrader UI, or for all assignments in a course by running a command in the integrated terminal. This functionality also satisfies the requirement that the grading process should be easy for the teacher to perform since the grading process is integrated into the existing Jupyter deployment.

5.3.3 Reliable grading process

The grading process should be reliable. If the submitted notebook contains malformed metadata or has other issues, the grading of that notebook will be skipped, and the teacher will be notified. In the proof of concept, the notification is implemented as a log message that is written to the logs of the grading pod. Each individual failed notebook produces their own error message in the log, with a summary of all errors at the end of the log. This provides the teacher with a detailed log of the grading process, including any errors that occurred during the grading. The log is accessible to the teacher by opening the log file in the filesystem via the JupyterLab interface. The process of finding and reading the log file is not as user-friendly as it could be, and there is room for improvement with the error reporting.

6 Conclusions

This thesis has created a proof of concept that fixes a long-standing problem: secure autograding with Nbgrader. The solution meets the three criteria set for the grading process, with the main focus on the first two: the solution is secure and allows self-service from the teachers' side. Additionally, it has aspects of the reliability requirement, and it is relatively easy to use since it is integrated into the existing Jupyter deployment. It is also possible to integrate the solution in other JupyterHub deployments.

The solution designed in this thesis is not yet fully integrated into the existing Jupyter cluster, but it was successfully tested in a local Kubernetes cluster. The solution was found to be functional, and the grading process completed without issues during trial runs.

The solution presented in Chapter 5 still requires some work before it is ready for production use. The solution is not yet completely integrated into the existing Jupyter deployment, and the mass-grading is not yet fully automated. In the proof of concept, only the public course data is being mounted to the grading pod. The volume mounts for the grading pods need to be improved so that they support dynamically mounting the necessary submission files alongside the public course data. In the future, a notification about a failed grading could be sent to the teacher's Jupyter session or displayed in the Formgrader UI, so that the teacher could inspect individual submissions in an interactive manner using the Formgrader UI.

The main goal of this thesis was to understand the possibilities of secure autograding and to create an initial proof of concept for running the grading in a secure, easy-to-use, and fast manner. The chosen method of implementing the grading process was to spawn dedicated grading pods in a Kubernetes cluster. A successful proof of concept was created, and the solution is suitable for further development and integration into the existing Jupyter deployment.

References

[1] Open Containers Initiative (OCI). *Open Container Initiative Runtime Specification*. Apr. 2024. URL: https://specs.opencontainers.org/runtime-spec/?v=v1.0.2.

[2] *Aalto JupyterHub for light computing and teaching*. URL: https://github.com/aaltoscicomp/jupyterhub-aalto (visited on 2024-02-14).

[3] *Apptainer*. URL: https://apptainer.org/docs/user/main/ (visited on 2024-04-25).

[4] The ZeroMQ authors. *ZeroMQ*. URL: https://zeromq.org/ (visited on 2024-02-14).

[5] Richard Darst and Alexander Ilin. *Jupyter for teaching: nbgrader / autograding*. CS-IT internal presentation. 2019.

[6] The pip developers. *pip documentation*. URL: https://pip.pypa.io/en/stable/ (visited on 2024-02-09).

[7] *envkernel – Switch environments before running Jupyter kernels*. URL: https://github.com/NordicHPC/envkernel (visited on 2024-02-18).

[8] Michael L. Epstein, Amber D. Lazarus, Tammy B. Calvano, Kelly A. Matthews, Rachel A. Hendel, Beth B. Epstein, and Gary M. Brosvic. "Immediate Feedback Assessment Technique Promotes Learning and Corrects Inaccurate first Responses". In: *The Psychological Record* 52.2 (Apr. 2002), pp. 187–201. ISSN: 2163-3452. DOI: 10.1007/BF03395423. URL: https://doi.org/10.1007/BF03395423.

[9] *Frequently Asked Questions*. URL: https://nbgrader.readthedocs.io/en/stable/user_guide/faq.html (visited on 2024-04-21).

[10] *Gateway Provisioners*. URL: https://gateway-provisioners.readthedocs.io/en/latest/ (visited on 2024-02-01).

[11] Cristian D. González-Carrillo, Felipe Restrepo-Calle, Jhon J. Ramírez-Echeverry, and Fabio A. González. "Automatic Grading Tool for Jupyter Notebooks in Artificial Intelligence Courses". In: *Sustainability* 13.21 (2021). ISSN: 2071-1050. DOI: 10.3390/su132112050. URL: https://www.mdpi.com/2071-1050/13/21/12050.

[12] *Grader Service*. URL: https://github.com/TU-Wien-dataLAB/Grader-Service (visited on 2024-01-17).

[13] Chris Greamo and Anup Ghosh. "Sandboxing and Virtualization: Modern Tools for Combating Malware". In: *IEEE Security and Privacy* 9.2 (2011), pp. 79–82. DOI: 10.1109/MSP.2011.36.

[14] Aalto CS-IT. *Jupyter Aalto Single-user image*. URL: https://github.com/AaltoSciComp/jupyter-aalto-singleuser/ (visited on 2024-02-09).

[15] Shashank Mohan Jain. "Cgroups". In: *Linux Containers and Virtualization: A Kernel Perspective*. Berkeley, CA: Apress, 2020, pp. 45–80. ISBN: 978-1-4842-6283-2. DOI: 10.1007/978-1-4842-6283-2_4. URL: https://doi.org/10.1007/978-1-4842-6283-2_4.

[16] Jeremiah W. Johnson. "Benefits and Pitfalls of Jupyter Notebooks in the Classroom". In: *Proceedings of the 21st Annual Conference on Information Technology Education*. SIGITE '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 32–37. ISBN: 9781450370455. DOI: 10.1145/3368308.3415397. URL: https://doi.org/10.1145/3368308.3415397.

[17] Project Jupyter. *Jupyter Docker Stacks*. URL: https://jupyter-docker-stacks.readthedocs.io/en/latest/ (visited on 2024-02-09).

[18] Project Jupyter. *JupyterHub Documentation*. URL: https://jupyterhub.readthedocs.io/en/stable/ (visited on 2024-02-14).

[19] Project Jupyter. *JupyterLab Documentation*. URL: https://jupyterlab.readthedocs.io/en/latest/ (visited on 2024-02-14).

[20] Project Jupyter. *Project Jupyter – About Us*. URL: https://jupyter.org/about (visited on 2024-02-07).

[21] Project Jupyter, Douglas Blank, David Bourgin, Alexander Brown, Matthias Bussonnier, Jonathan Frederic, Brian Granger, Thomas Griffiths, Jessica Hamrick, Kyle Kelley, M Pacer, Logan Page, Fernando Pérez, Benjamin Ragan-Kelley, Jordan Suchow, and Carol Willing. "nbgrader: A Tool for Creating and Grading Assignments in the Jupyter Notebook". In: *Journal of Open Source Education* 2.16 (2019), p. 32. DOI: 10.21105/jose.00032. URL: https://doi.org/10.21105/jose.00032.

[22] Ville Karavirta, Petri Ihantola, and Teemu Koskinen. "Service-Oriented Approach to Improve Interoperability of E-Learning Systems". In: *2013 IEEE 13th International Conference on Advanced Learning Technologies*. 2013, pp. 341–345. DOI: 10.1109/ICALT.2013.105.

[23] *Kubespawner – profile_list*. URL: https://jupyterhub-kubespawner.readthedocs.io/en/latest/spawner.html#kubespawner.KubeSpawner.profile_list (visited on 2024-02-14).

[24] *Lmod – A New Environment Module System*. URL: https://lmod.readthedocs.io/en/latest/ (visited on 2024-01-15).

[25] Hamza Manzoor, Amit Naik, Clifford A. Shaffer, Chris North, and Stephen H. Edwards. "Auto-Grading Jupyter Notebooks". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE '20. Portland, OR, USA: Association for Computing Machinery, 2020, pp. 1139–1144. ISBN: 9781450367936. DOI: 10.1145/3328778.3366947. URL: https://doi.org/10.1145/3328778.3366947.

[26] *MOOC-Grader – Automatic assessment framework compatible with A-plus LMS*. URL: https://github.com/apluslms/mooc-grader (visited on 2024-01-17).

[27] Adrian Mouat. *Using Docker: Developing and deploying software with containers*. " O'Reilly Media, Inc.", 2015.

[28] Ruchika Muddinagiri, Shubham Ambavane, and Simran Bayas. "Self-Hosted Kubernetes: Deploying Docker Containers Locally With Minikube". In: *2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET)*. 2019, pp. 239–243. DOI: 10.1109/ICITAET47105.2019.9170208.

[29] Jérôme Petazzoni. *Do not use Docker-in-Docker for CI*. URL: https://jpetazzo.github.io/2015/09/03/do-not-use-docker-in-docker-for-ci/ (visited on 2024-04-21).

[30] David Rensin. *Kubernetes – Scheduling the Future at Cloud Scale*. O'Reilly Media, Incorporated, 2015. ISBN: 978-1-491-93599-6.

[31] Gigi Sayfan. *Mastering kubernetes*. Packt Publishing Ltd, 2017. ISBN: 9781786469854.

[32] Anuj Ramesh Shah. "Web-cat: A web-based center for automated testing". PhD thesis. Virginia Tech, 2003.

[33] Madiha H Syed, Eduardo B Fernandez, et al. "The software container pattern". In: *Proceedings of the 22nd Conference on Pattern Languages of Programs*. 2015, pp. 24–26.

[34] Jupyter Team. *Installing the Classic Jupyter Notebook interface*. URL: https://docs.jupyter.org/en/latest/install/notebook-classic.html (visited on 2024-02-07).

[35] Jupyter Development Team. *Grading in a docker container – nbgrader documentation*. URL: https://nbgrader.readthedocs.io/en/stable/user_guide/advanced.html#grading-in-a-docker-container (visited on 2024-02-15).

[36] Jupyter Development Team. *Messaging in Jupyter*. URL: https://jupyter-protocol.readthedocs.io/en/latest/messaging.html (visited on 2024-02-14).

[37] Jupyter Development Team. *The Jupyter Notebook Format*. URL: https://nbformat.readthedocs.io/en/latest/ (visited on 2024-01-31).

[38] *The Littlest JupyterHub*. URL: https://tljh.jupyter.org/en/latest/ (visited on 2024-02-14).

[39] *Zero to JupyterHub with Kubernetes*. URL: https://z2jh.jupyter.org/en/stable/ (visited on 2024-02-14).