Architectural Kata Design Solution & Report:

Check Your Work

Class	CS 7346
Semester	Spring 2025
Student	Davis Vercher
SMU ID	49377022

Table of Contents:

I. Abstract

AutoTA, an automated simple programming assignment grading and plagiarism detection system, is proposed as a solution to the “Check Your Work” architectural kata [1]. AutoTA involves API integration with the Canvas learning management system (LMS) for a notional university, and an Amazon Web Services (AWS)-based execution, grading, storage, and plagiarism detection pipeline. AutoTA provides the university with a secure and scalable automated grading solution for its growing Computer Science department – handling the grading of simple student programing assignments and enabling Professors and Teaching Assistants (TAs) to spend less time grading and more time delivering high-quality teaching and mentorship to the university’s students.

II. Introduction

The Architectural Kata for this project is “Check Your Work.” This kata outlines a scenario where a university (Southern Computer University – SCU) wants to build an automated grading system for simple programming assignments in order to meet the demand of a greatly expanded Computer Science department [1]. The implementation and roll-out of AutoTA for SCU’s Computer Science department will be piloted for one semester and evaluated to see if AutoTA is a viable solution for the University’s goal of reducing overall costs by reducing the number of TAs required to be employed by the Computer Science department. If AutoTA can reliably be used for the grading of simple programming assignments for the nearly 300 students in introductory-level Computer Science courses (the grading of which requires time-intensive manual work from several TAs per course), and it can be done at a cost that is less than the cost of employment for real TAs, then the University plans on utilizing AutoTA permanently.

This project to automate simple assignment grading for programming homework assignments will reduce the burden of grading simple, runnable code for a large group of students by the Professors and TAs. Furthermore, Check Your Work outlines more specific requirements for this kata. The number of users will be nearly 300 per year, and will include students, staff, and administrators. Individual students must be able to upload their source code as homework assignments within the university’s Canvas LMS before the cloud solution runs and grades the submission. Additionally, grades and code runs must be both persistent and auditable, and a plagiarism detection system must be implemented to check the individual homework submission against all other submissions within the cloud system and via integration with a web-based plagiarism system like Turnitin [1].

III. Overview of the Solution

To meet these requirements, I propose the following solution – a project titled "AutoTA." AutoTA will use AWS as the cloud service provider for this project, providing all compute, storage, and integration (with Canvas). AutoTA's grading pipeline, at a high-level, will consist of the following: 1) submission of assignment via Canvas LMS (in both code file and duplicate .html file); 2) transfer of the files to AWS S3 via AWS API Gateway (with validation of file via AWS Lambda function and the start of logging via AWS CloudTrail); 3) queueing of the submitted code file via AWS Simple Queue Service (SQS); 4) execution of the code file inside an AWS Fargate instance; 5) grading of correctness of the submitted code against the Professor's solution (expected outputs of the code); 6) storage of assignment performance/grade information in AWS DynamoDB; 7) comparison of submitted code file with previously submitted assignment files already stored in S3 via Amazon Athena; 8) API submission of .html file to Turnitin for comparison, with results returned via API and stored in AWS DynamoDB; 9) API push of assignment grade, internal plagiarism check, and Turnitin plagiarism check results back to Canvas LMS. Steps 3 through 9 of the pipeline will be orchestrated by AWS Step Functions.

IV. Assumptions

Several key assumptions were made when designing the AutoTA system (in relation to the requirements from the CYW kata) – specifically relating to the student assignment submission files (programming languages, formatting, etc.), assignment rubrics (how they are actually graded), SCU budget for implementing AutoTA, what is done if plagiarism is detected, user authentication, user interactions with AutoTA, and other various items.

With regard to the files that are submittable for the AutoTA system to function properly, there are two considerations – one for students uploading assignments and one for Professors/TAs uploading "answer key" solution files. As a first basic assumption, it is assumed that Professors/TAs will clearly communicate to their students the expectation for naming conventions of the submitted files (i.e., “CS7346_spring2024_assignment1_firstname_lastname.py” as an example naming structure), and that this naming convention will be enforced as a rule in the course of the class itself, and not by AutoTA. Similarly, it is assumed that for any given assignment, the Professor or TA will upload the solution file to Canvas before the assignment is open for students to submit their code/html files, and that these files will adhere to a specific naming convention that AutoTA will recognize (i.e., “CS7346_spring2024_assignment1_solution.py”). Additionally, it is assumed that only one solution file for any particular assignment can exist in AutoTA, and if a Professor/TA uploads a new solution file (bearing the exact same name as previously uploaded files) that AutoTA will automatically overwrite the existing file in S3. This means that it is the responsibility of the Professor/TA to not update the solution file during a period of time that students are able to upload their own homework files for a particular assignment. It is also assumed that the programming assignments themselves will have strictly defined requirements for the students to implement, such as defining basic functions (and not constructing complex classes or multi-file workflows). Student submission files will operate in a similar manner to solution file submissions – allowing for students to resubmit their homework files multiple times during the period that they are allowed to do so in Canvas, and their most recent upload will be the one graded by AutoTA. Next, it is assumed that Professors/TAs will instruct their students to only submit main source code file for assignments, meaning that a C++ assignment will have one ‘.c’ or ‘.cpp’ file as the submission (along with .html copy) and will not have the ability to handle a submission with headers separated out into a ‘.h’ or ‘.hpp’ file. Furthermore, it is assumed that the only valid programming languages handled by AutoTA are Python, C++, and Java, and the coding assignments are simple enough in nature to only encompass work with simple input/outputs, such as defining a square root function that outputs 4 with an input of 16.

With regard to grading assignments, it is assumed that the student code files will be executed by AutoTA with specific inputs for specific functions to evaluate correctness. These functions and inputs will be provided by the Professor’s solution file. The final grade for an assignment will be the percentage of correct outputs of the student’s code when compared to the solution outputs, rounded to the nearest percentage point. For example, if a solution to an assignment has 10 expected outputs, and a student's code outputs 6 of the 10 validly, the grade will be 60%.

With regard to plagiarism detection, if AutoTA flags a student's submission as possibly containing plagiarism (either through internal AutoTA check or via Turnitin.com check), it is assumed that the Professor will be responsible for actually investigating further to determine if plagiarism is in fact present and what disciplinary-type actions will be taken against the student. AutoTA should not serve as the true judgement of plagiarism, but rather as a plagiarism indicator/detector.

With regard to other minor assumptions, the following is assumed for this project: 1) Professors can manually override a student's grade from AutoTA if needed (in case of AutoTA error); 2) AutoTA does not handle user authentication – meaning if a user submits files in Canvas it is assumed the user has a valid ability to do so; 3) AutoTA does not enforce deadlines for submissions, since this is handled in Canvas; 4) AutoTA will not be interactable by university staff/Professors, as it does not have a graphical user interface (GUI) or command line interface (CLI) for general system users (e.g., students, Professors, University leadership); 5) SCU has the budget required to stand-up and maintain AutoTA; 6) AutoTA will be evaluated by SCU's legal team before build and launch to ensure there are not legal data privacy issues; and 7) SCU does not use AWS for any other purpose, and therefore the system administrator users (technical maintainers of the system) will have a root user role in AWS.

V. System Actors

Beginning with the actors using AutoTA, there are three main categories: 1) end users, or file submitters (Professors/TAs and students); 2) system administrators; and 3) stakeholders, or SCU administration leadership. Category 1 (end users) are necessarily sub-divided into two types – Professors/TAs who use AutoTA to grade student programming assignments, and students who have their assignments graded via AutoTA.

End users are necessarily split into two sub-categories (Professors/TAs and students), but both types of end-users interact with the system in similar ways – primarily through file upload in Canvas. Where the Professor/TA end users differ from the students, though, is that this type of end user has the additional interaction with AutoTA of receiving email notifications when assignments are flagged for plagiarism by AutoTA.

System administrators are the maintainers of the AutoTA system and necessarily will need to have robust knowledge of AWS in general, as well as detailed knowledge of the specific AWS components of AutoTA. These users will have access to AutoTA via Identity Access Management (IAM)/root user roles in the AWS console (GUI) and AWS CLI, and will be responsible for maintaining the AutoTA system and providing reports to SCU leadership about AutoTA (e.g., monthly cost reports, usage reports, etc.).

Stakeholder users, such as the SCU administrator leadership (i.e., Computer Science Department Chair) will not have direct interactions with AutoTA, but will receive reports from AutoTA system administrators and will be responsible for providing the budget for AutoTA's creation and ongoing maintenance. These users will likely interact with AutoTA via live demonstrations of the system in action during its creation process.

VI. System Use Cases

As discussed previously, the primary use case of AutoTA will be for the grading and plagiarism checking of simple programming assignments submitted in Canvas by SCU Computer Science students. This is the core functionality of the system and the purpose it is built for. However, another ancillary purpose of AutoTA is to serve as a cloud repository of SCU's basic programming assignments from its Computer Science department. After a few semesters of AutoTA's use, the university has the ability to look back at the assignments from previous semesters for reference – especially useful to determine if basic programming assignments are trending up or down in complexity over time. Additionally, this repository use case of AutoTA will allow the Computer Science department leadership to audit these simple programming assignments to ensure that the assignments are remaining relevant to specific Computer Science courses (i.e., an "Intro to Python Programming" course does not have Java or C++ assignments).

VII. Solution Details

When a student submits a coding homework assignment suitable for AutoTA, they do so as a correctly formatted source code file relative to the coding language of the homework (e.g., '.py' for Python files, '.cpp' for C++, & '.java' for Java) – ensuring that their files are less than 5 GB (the Canvas file upload limit [2]). Additionally, at the same time they will be required to submit a '.html' version of the code file, which the AutoTA pipeline will later use for plagiarism checking with Turnitin.com (which supports '.html' files up to 100 MB in size [3]). After the submission is uploaded in Canvas, the Canvas LMS system forwards the submitted file by calling an AWS API Gateway (authenticating via OAuth2 [4]) endpoint, which triggers an AWS Lambda function that validates the submission (file size, file type, student authorization, etc.), tags the specific files (with things like the student ID and assignment ID), and generates a pre-signed S3 URL – allowing the files to be uploaded into an AWS S3 bucket and retrievable later in the process. Once the submitted files have made their way into S3, AutoTA notifies Canvas via API response that the files have been successfully received into the AutoTA grading pipeline. Additionally, AWS CloudTrail will begin logging operations on the submission pipeline process to ensure trackability.

After receipt into S3, a Lambda function copies the code file and converts it into structured plaintext CSV file – this will be used later in the pipeline for internal plagiarism checks. Next, AWS Simple Queue Service (SQS) is used to manage all assignments submitted into S3 before they are picked up by AWS Fargate – a serverless compute engine that works with AWS Elastic Container Service (ECS) to provide secure and isolated execution environments with no public access [5] (for running untrustworthy student submitted code) – for code execution (can also work with EKS but not used here). The use of SQS will allow for multiple students to submit assignments in Canvas without exhausting the resources for the AutoTA pipeline, due to SQS queueing submissions for execution in the pipeline when resources are available (i.e., in times of extreme demand).

Asynchronously from this pipeline process and before assignment can be graded, the Professor for a particular assignment will be required to upload a file via Canvas with the expected outputs of the student code in a file format that matches the assignment (i.e., 'solution.py'). This file will flow into the AutoTA pipeline the same as a student homework submission and will be tagged with an assignment ID that matches the student submissions for that particular assignment.

Once the student’s code file is executed in a Fargate instance, the outputs of the code are checked against the professor’s provided solution. This is done via Step Functions retrieving the expected answers (professor’s solution file) from S3, a Lambda function comparing the output against the answer, and the calculation of the percentage of correctness of the student’s executed code. This percentage of correctness serves as the assignment grade and is then recorded in DynamoDB for later retrieval and for future auditability.

After the grade is recorded in the pipeline, AutoTA then checks the CSV copy of the code file against previously submitted assignments stored in the system’s S3 bucket (via Athena query) to determine if the submitted assignment meets the threshold of 70% similarity to other submitted assignments. This is done via Athena queries of the extracted code text in S3. If the assignment meets this threshold, Step Functions trigger a Lambda function to update the assignment ID record with a plagiarism flag in DynamoDB. Next, a Lambda function takes the ‘.html’ version of the assignment code and triggers an API call to Turnitin.com for further plagiarism detection. Turnitin.com then returns the plagiarism report via API to AutoTA, and if the report indicates valid plagiarism, then a plagiarism flag is attached to the assignment in DynamoDB. If a plagiarism flag is attached by either detection system (internal and Turnitin.com), then AWS Simple Notification Service (SNS) pushes an email alert to the Professor and appropriate TA of the course with the assignment and student name/student ID of the potential plagiarism (so the Professor/TA can investigate further).

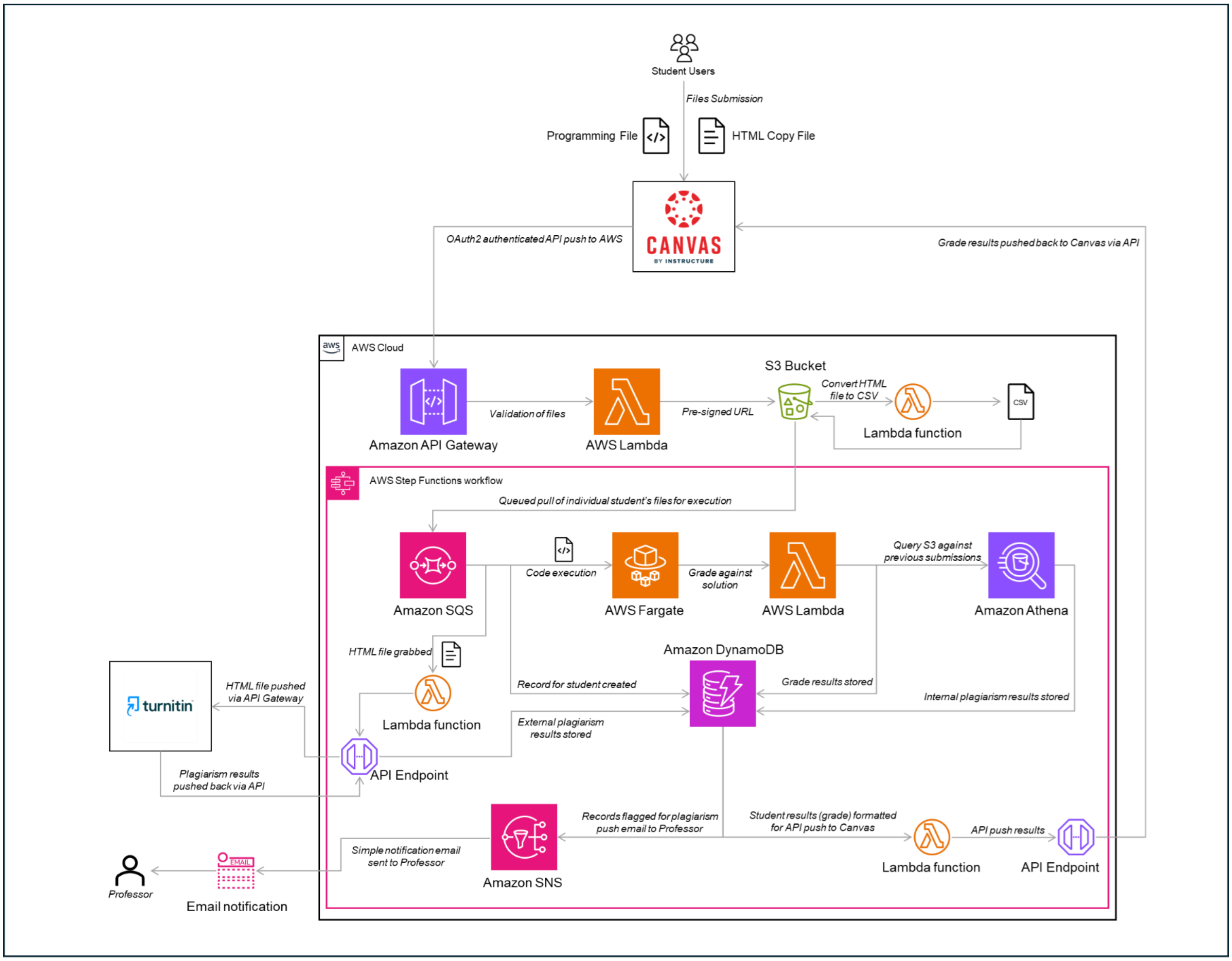
Finally, Step Functions aggregate the grade results and plagiarism findings, and a Lambda function formats the assignment data before returning the information back to Canvas via API. When Canvas confirms receipt of the information, CloudTrail logs the confirmation response for the assignment.

Outside of the scope of the actual pipeline, AutoTA will also enable system administrators to access cost and usage reports via AWS CloudWatch dashboards (usage statistics) and AWS Cloud Explorer for historical cost data and future cost trend predictions [6].

These reports will be presented to SCU leadership on a monthly and quarterly basis to enable Stakeholder users to monitor AutoTA's cost and use and make program decisions related to AutoTA's return on investment.

VIII. Solution Diagram

See below for a diagram of the AutoTA system (Figure 1 – Solution Diagram):



Student Users
Files Submission
Programming File
HTML Copy File
CANVAS BY INSTRUCTURE
OAuth2 authenticated API push to AWS
Grade results pushed back to Canvas via API
AWS Cloud
Amazon API Gateway
Validation of files
AWS Lambda
Pre-signed URL
S3 Bucket
Convert HTML file to CSV
Lambda function
CSV
AWS Step Functions workflow
Queued pull of individual student's files for execution
Amazon SQS
Code execution
AWS Fargate
Grade against solution
AWS Lambda
Query S3 against previous submissions
Amazon Athena
HTML file grabbed
Lambda function
Amazon DynamoDB
Record for student created
Grade results stored
Internal plagiarism results stored
External plagiarism results stored
turnitin
HTML file pushed via API Gateway
API Endpoint
Plagiarism results pushed back via API
Records flagged for plagiarism push email to Professor
Student results (grade) formatted for API push to Canvas
API push results
Lambda function
API Endpoint
Amazon SNS
Simple notification email sent to Professor
EMAIL
Email notification
Professor

Figure 1. AutoTA Solution Diagram

Davis Vercher - 49377022

IX. Solution Priority Discussion

Because SCU's goal in implementing AutoTA is to reduce Computer Science department costs by lowering TA headcount, affordability and reliability are top priorities. First, if AutoTA isn't reliable at grading simple programming assignments or checking for plagiarism, TAs will still be needed—nullifying its value. Second, if the total cost of deploying and maintaining AutoTA isn't realistically lower than hiring TAs, using it would not be financially justified. This would matter less if the goal of AutoTA was primarily for research rather than cost savings.

Security is also a priority due to AutoTA's function of executing unverified student code. Students learning programming are very likely to introduce errors that could impact cloud resource usage. While benign issues like infinite loops can be handled separately, the real concern is malicious code submitted to exploit AutoTA and possibly gain access to University systems – for example altering payroll systems. Mitigations for this risk are covered in Section XI.

As a trade-off, availability, performance, and usability are de-emphasized. AutoTA needs to be accessible for assignment submissions but doesn't need to process them instantly. Since human-graded assignments typically take 2–7 days for feedback, AutoTA only needs to at the very least match this timeline. With about 300 users, performance demands remain relatively low compared to systems with massive scale (i.e., 100,000+ simultaneous requests).

Usability is a lower priority due to how users interact with AutoTA—primarily through Canvas. Most users (students and faculty) only submit files and receive grades, requiring no direct interaction with AutoTA. System administrators will use AWS Console and CLI to manage it, making a dedicated GUI or CLI interface unnecessary.

X. Solution Component Discussion

Overall, the choice to use AWS for all components of AutoTA in a public cloud environment is mainly for the ease of development and maintenance. Because AutoTA is being developed for a pilot launch for SCU – a proof of concept – AutoTA's existence as an AWS-only entity eases development time and ongoing maintenance work by the System Administrators. If AutoTA was built utilizing disparate components from different cloud service providers (CSPs) such as Microsoft Azure or Google Cloud Platform, then the effort required to implement the system would be dramatically increased due to the need for integration of pieces from different CSPs (likely requiring some form of custom-built software to handle the integrations). AWS was chosen over other CSPs due to the fact that this course (Dr. El Dayeh's CS7346 – Spring of 2025) is primarily focused on AWS, and this course is my first introduction to studying CSPs.Each AWS component chosen for AutoTA was selected for specific reasons. For instance, Amazon S3 was picked as the file repository over Amazon EFS because AutoTA stores simple programming assignments that don't require real-time shared file access. Files are only accessed by one Fargate instance during execution, making S3 a better fit.

AWS SQS was selected to queue submitted files for execution, as it's more cost-effective than Amazon Kinesis. While Kinesis could offer faster execution performance, SQS better aligns with AutoTA's goal of affordability over speed.

AWS Fargate was chosen to run student code in a sandboxed environment instead of AWS Lambda. Fargate supports C++, which Lambda does not, and it's serverless nature reduces infrastructure management for System Administrators. More importantly, Fargate provides a safe, reliable way to run potentially malicious student code.

Amazon Athena was selected for querying S3-stored files with SQL, because it can directly read CSV-formatted data in S3. Unlike AWS Redshift, which is designed for complex queries, Athena is a simpler and more suitable tool for AutoTA's needs.

Finally, AWS SNS was chosen over AWS SES to send plagiarism alerts to professors. SNS avoids the need to manage email templates, making it a easier to use and a more efficient way to notify instructors about potential plagiarism.

XI. Risks & Mitigations

Discussed above in Section IX, a main risk from AutoTA's structure is in the execution of untested student code. If an attacker was able to gain access to Canvas and upload a malicious file that was ingested into AutoTA's pipeline, this code would be executed and could do harm to the system (i.e., allowing for unauthorized sensitive data access). To mitigate this threat, three mitigations are built into the AutoTA system.

First, AutoTA will exist in its own AWS space as a standalone role. This means that there will be no accessible connections between the resources of the AutoTA AWS role and any other existing SCU AWS roles/uses. This will prevent malicious code injection in AutoTA from somehow allowing for access to SCU's broader enterprise network (that may be utilizing AWS, i.e., for University employee payroll information or Personally Identifiable Information (PII) such as social security numbers). System Administrator users of AutoTA (with access/control within AWS for AutoTA) will be University employees and/or contractors that are vetted before being given access to the system (and potentially liable for security breaches). Additionally, these System Administrators will be required to have operational practices put into place to ensure their own access security (IAM user roles/hierarchies, multi-factor authentication (MFA), etc.).

Second, file submission into AutoTA will only be available through Canvas by Canvas users that have the permission to be in a course utilizing AutoTA for assignment grading. This means that the only people with access to file submission have been vetted by the University itself, and if malicious code is intentionally uploaded, the source of the code is at least traceable to individuals that are known entities to the University (Professors, TAs, and students). Though this mitigation "off-shores" access management to Canvas, SCU has in-place security measures already for Canvas access (i.e., MFA login to Canvas via an application like Cisco's Duo service [7]) that AutoTA can de facto utilize to mitigate this risk.

Finally, the use of AWS Fargate as the container to actually execute student code will compartmentalize the execution of the code away from the rest of the AutoTA system in a sandbox environment, greatly limiting the impact of the execution of malicious code (discussed during Section X). Though Fargate does not have a hard run-time limit of 15 minutes like Lambda, the system will be configured in a way to stop Fargate execution after only a few minutes for the simple student code (in the case of infinite loops).

XII. Limitations of Cloud Services Chosen

Choosing an all-AWS solution for the AutoTA poses a few limitations to consider. AutoTA will be tightly coupled to the AWS ecosystem and could fall into being locked in with AWS because the effort and cost required to migrate to another CSP could be too costly, effectively trapping the AutoTA data in AWS. Additionally, AutoTA will heavily rely on knowledgeable and skilled AWS professionals to be the System Administrators, requiring SCU to maintain a group of these kinds of employees (or contracts with these kinds of consultants/contractors) which will grow in cost over time.

XIII. Conclusion

AutoTA is a proposed solution to the CYW architectural kata through the use of AWS submission, grading, plagiarism detection, and return pipeline integrated into a university's existing Canvas LMS and is focused on reliability, affordability, and security over performance/availability and usability.

An important note on the limitations of the design of AutoTA: I understand that the design decisions I have chosen for AutoTA likely include errors in the choice of AWS component and how these components operate together to form a cohesive and usable pipeline. Since this course is my first exposure to cloud computing (and AWS) I expect that there have been errors made in AutoTA's design that I will learn from. However, I have done my best to ensure that the design of AutoTA is logical and maps onto the reality of how the AWS components should integrate with each other.

Looking ahead, though AutoTA is a notional system for a fake university, it has the potential to evolve into a more realistic project actually usable by real universities. Making this transition would require much more time spent understanding the components of the system, trade-offs, and identifying opportunities for efficiency enhancements. Additionally, it would be very interesting to study how the AutoTA system could be adapted to grade more than just simple programming assignments, such as with the integration of code-focused Large Language Model (LLM) generative AI tools.

XIV. References

[1] “Check Your Work,” *architecturalkatas.com*, 2012. https://www.architecturalkatas.com/kata.html?kata=CheckYourWork.json

[2] “How do I upload a file as an assignment submission in Canvas?,” community.canvaslms.com, Jul. 20, 2020. https://community.canvaslms.com/t5/Student-Guide/How-do-I-upload-a-file-as-an-assignment-submission-in-Canvas/ta-p/274

[3] “FAQ,” Turnitin.com, 2025. https://developers.turnitin.com/turnitin-core-api/faq

[4] “OAuth2 - Canvas LMS REST API Documentation,” canvas.instructure.com. https://canvas.instructure.com/doc/api/file.oauth.html

[5] “AWS Fargate - Run containers without having to manage servers or clusters,” *Amazon Web Services, Inc.*, 2025. https://aws.amazon.com/fargate/

[6] AWS, “AWS Cost Explorer - Amazon Web Services,” *Amazon Web Services, Inc.*, 2025. https://aws.amazon.com/aws-cost-management/aws-cost-explorer/

[7] “Two-factor authentication (2FA),” Cisco Duo, https://duo.com/product/multi-factor-authentication-mfa/two-factor-authentication-2fa?utm_source=google&utm_medium=paid_search&utm_campaign=DUO_AMER_NA_GS_Branded_General_T1&utm_content=APP&_bn=g&_bk=duo+authenticator&_bm=e&_bt=659258609488&_bg=146748779717&gad_source=1&gclid=Cj0KCQjw7dm-BhCoARIsALFk4v_ZgJc5wJNWeiimPcLntu26K4DurBevHzAk94nUoeJcJ_OpRsC0-moaAjbKEALw_wcB

[8] AWS, “Amazon Kinesis,” *Amazon Web Services, Inc.*, 2025. https://aws.amazon.com/kinesis/

[9] AWS, “Amazon Redshift,” *Amazon Web Services, Inc.*, 2025. https://aws.amazon.com/pm/redshift/

[10] AWS, “Amazon Simple Email Service (Amazon SES),” *Amazon Web Services, Inc.*, 2025. https://aws.amazon.com/ses/