# The AutoTA System

*“Check Your Work” Kata Solution*

**Davis Vercher**
Final Presentation
CS7346
Spring 2025

## Check Your Work | Kata Overview

Link to the Kata

*Kata Background*

A university has greatly expanded its CS department and wants to be able to **automate the grading of simple programming assignments**.

**Users**: 300+ students per year, plus staff and admin

*Kata Requirements*

Students must be able to **upload their source code**, which will run and be **graded automatically.**

The solution must have:

| ***Persistent & Auditable*** | ***Plagiarism Detection*** | ***Integration*** |
|---|---|---|
| …grade runs that exist within the solution system | …via both internal checks against previous submissions in the system & via external Turnitin.com checks | …with the university's Learning Management System (LMS), i.e., Canvas |

## AutoTA | Kata Solution & Additional Context

*Solution Overview:*

**"AutoTA"** is the proposed solution for the kata, and is an **AWS-based** grading, plagiarism detection, and storage pipeline fully integrated with the university's Canvas LMS.

*Scenario Information:*

Southern Computer University (SCU) is **piloting** AutoTA for **one semester**, to determine if the system can lower overhead costs by allowing them to reduce Teaching Assistant (TA) headcount.

If AutoTA can **reliably** and **affordably** grade all simple programming assignments from the CS Department of SCU, then SCU will move to reduce the numbers of real TAs in the CS Department and **use AutoTA permanently**.

## AutoTA | High-Level Process

### 1 | *File Submission in Canvas*

Student submits code into Canvas via normal assignment upload process (one source code file and one copy HTML file)

### 2 | *Ingestion into AutoTA*

File submission in Canvas is trigger for AutoTA to ingest submitted files via API into AWS

### 3 | *Code Execution & Grading*

AutoTA executes code in sandbox and checks against Professor's previously uploaded solution

### 4 | *Plagiarism Checks*

Internal & external plagiarism checks are conducted (within AWS and via API to Turnitin.com)

### 5 | *Grade Returned to Canvas*

AutoTA returns grade to Canvas

If AutoTA detects possible plagiarism, an email is sent by AutoTA to Professor

# Assumptions

## *File Submissions*

- Files will be named in a format AutoTA expects
- Professors enforce naming conventions on files
- Only allowable languages are Python, Java, and C++
- Student file submissions in Canvas overwrite previous submissions ("Multiple Attempts")
- Only 1 main code source file per assignment (i.e., no header files for C++ program), along with copy of the file in HTML format (used for plagiarism detection)

## *Grading*

- Professors upload solution file to AutoTA (with specific 'solution' naming) before allowing students to submit in Canvas
- Student code is graded by comparing executed code outputs to solution outputs - % correct is the grade (i.e., 6 out of 10 correct outputs = 60% grade)

## *Plagiarism Detection*

- AutoTA only flags for potential plagiarism
- Professor's are responsible for investigating potential plagiarism to make final determination (Human control)

## *Other*

- Professors can override grades in Canvas if desired
- AutoTA does not handle user authentication for users submitting files (handled by Canvas – AutoTA assumes anyone submitting files is authorized to do so)
- Deadlines for assignments not enforced by AutoTA – handled by Canvas deadlines
- SCU has budget required for the pilot of AutoTA
- SCU's legal team has granted AutoTA legal approval for the pilot

## Actors

| *End Users* (File Submitters) | *System Administrators* (AWS Practitioners) | *Stakeholders* (SCU Leadership) |
|---|---|---|
| • **Professors**<br>• Upload solution files, receive plagiarism notifications<br>• **Only direct interaction with AutoTA is via plagiarism notification emails**<br><br>• **Students**<br>• Submit assignments for grading<br>• **No direct interaction with AutoTA** | • **Contracted and/or Hired AWS Team**<br>• Build AutoTA system for pilot<br>• Provide ongoing maintenance/operation services on the system<br>• **Have access to the system via AWS Console and AWS CLI (high interaction)**<br>• Build cost/usage/performance reports for Stakeholders<br>• Help with final analysis of AutoTA pilot | • **SCU & Department Leadership**<br>• Oversee AutoTA program<br>• Receive ongoing reports from System Administrators<br>• Make final determination on success of AutoTA pilot<br>• Provide funding for AutoTA<br>• **Only indirect interaction with AutoTA via product demos and system reports** |

## Use Cases

There is one main use case and one ancillary use case of AutoTA:

- **Main**: Automated simple programming assignments grading and plagiarism detection
- **Ancillary**: Historical repository of the assignments

### *Main Use Case*
(Subject of this presentation)

- **Description**
  - Integrate with SCU's Canvas LMS to allow for automated grading and plagiarism detection for simple programming assignments (CS Department)
- **Purpose**
  - Decrease time needed to grade assignments that are tedious and time-intensive

### *Ancillary Use Case*
(Not covered in depth in this presentation)

- **Description**
  - Serve as a repository of all past semester CS Department simple programming assignments (across all basic CS courses)
  - Allow for previous simple programming assignments (with solutions) to be retrievable for future review/use
- **Purpose**
  - Allow for future course analysis of basic CS courses at SCU by Department/University leadership
  - Track simple programming assignment complexity over time (course/assignment fit)
  - Ensure course/assignment alignment (i.e., only Python assignments in a Python-focused CS course)

## Priorities

*Prioritized*

**AFFORDABILITY**

**RELIABILITY**

**SECURITY**

*Justification:*

AutoTA must prove itself to be able to perform the task of grading simple programming assignments at least as reliably as real TAs (**reliability**), while costing SCU less than the equivalent amount of TA work (**affordability**).

Additionally, AutoTA must mitigate the risk of malicious code injection from uploaded student files (**security**) – discussed later in the presentation.

*De-prioritized*

**PERFORMANCE**

**USABILITY**

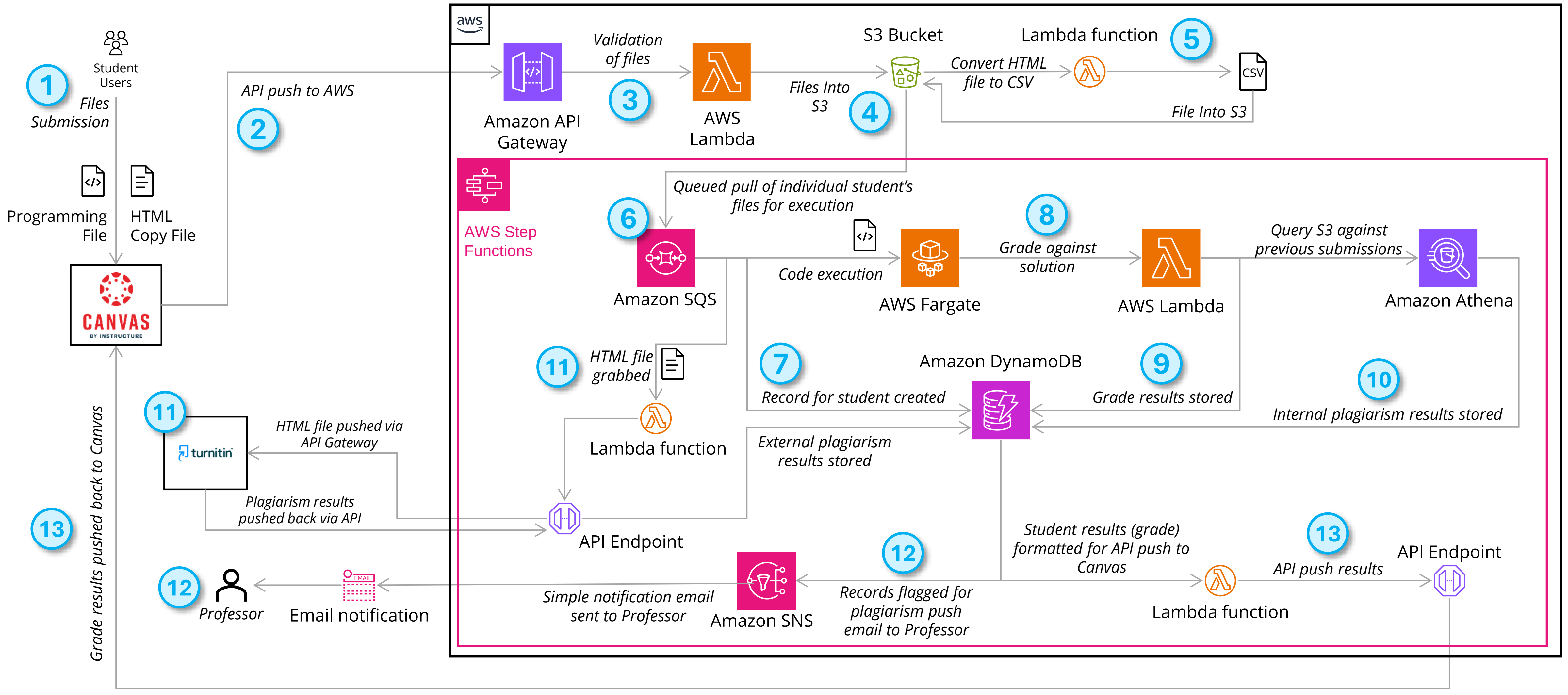
*Justification:*

AutoTA processing of student files does not need to be real-time, as the benchmark for real TA grading is roughly 2-7 work days (**performance**).

AutoTA, especially for the pilot phase, does not need to prioritize usability for file submitter end-users (no GUI tool) – interactions for these users are indirect and through Canvas (**usability**).

# Architecture

**Key:** X Step number for individual student submission within AutoTA



## Solution Components

*Why AWS?*

Overall, the choice of a full-AWS solution for AutoTA in a public cloud environment is for **ease of development and maintenance** for the pilot phase.

Utilizing just AWS for the pilot phase of AWS (proof of concept) **avoids lengthy and complex integrations** of a **multi-cloud** environment (i.e., avoiding custom-built software to handle integration of multiple CSP cloud services)

*AWS Component Decisions with Reasoning:*

| **S3**<br>***File Storage*** | **SQS**<br>***Submission Queueing*** | **Fargate**<br>***Sandbox Execution*** | **Athena**<br>***Record Querying*** | **SNS**<br>***Email Alerts*** |
|---|---|---|---|---|
| Simple and effective for use storing student programming files (and copies in other formats) | Queuing of submissions in AutoTA pipeline does not need to handle many concurrent items (low throughput) | Sandboxed environment to run untested student code – low requirement to manage servers manually | CSV copies of student code need to be queried for internal plagiarism checks, but these queries will be simple | Professor's will receive emails in event of possible plagiarism. SNS provides very simple email alerts. |
| *Alternative:*<br>***AWS Elastic File System***<br>Real-time sharing of the files between instances/people not needed | *Alternative:*<br>***AWS Kinesis***<br>Higher performance, but higher cost – affordability prioritized over performance | *Alternative:*<br>***Standard AWS EC2***<br>EC2 would require more manual server management for executing student code | *Alternative:*<br>***Amazon Redshift***<br>Athena allows for simple querying in S3 – Redshift is meant for much more complex querying | *Alternative:*<br>***AWS Simple Email Service***<br>Would provide same service, but would require creation/management of email templates |

## Risks & Mitigations

*Risk Matrix*

| | | Impact | | | | |
|---|---|---|---|---|---|---|
| | | Insignificant (1) | Minor (2) | Moderate (3) | Major (4) | Catastrophic (5) |
| Probability | Very Likely (5) | Medium 3 | High 4 | High 4 | Very High 5 | Extreme 6 |
| | Likely (4) | Medium 3 | Medium 3 | High 4 | High 4 | Very High 5 |
| | Occasional (3) | Low 2 | Medium 3 | Medium 3 | High 4 | High 4 |
| | Unlikely (2) | Low 2 | Low 2 | Medium 3 | Medium 3 | High 4 |
| | Rare (1) | Very Low 1 | Low 2 | Low 2 | Medium 3 | Medium 3 |

*Risk A:* **Malicious Code Injection via Student File**

| | |
|---|---|
| *Description* | Intentional injection of malicious code via student file upload. AutoTA executes student code as part of normal pipeline, malicious code takes advantage of this process for malicious purposes (i.e., gaining access to broader SCU network) |
| *Probability* | Unlikely (2) |
| *Impact* | Catastrophic (5) |
| *Risk Assessment* | High (4) |
| *Mitigation Plan* | • Standalone AWS role (separate from broader SCU IT infrastructure)<br>• Canvas-based access (only known users can submit files) – helps with tracking in event of malicious code injection<br>• Use of AWS Fargate for code execution (isolated sandbox environment)<br>• System Administrators (with AWS Console access) will adhere to operational best practices (e.g., MFA, IAM user roles/permissions, etc.)<br>• CloudWatch logging for post-incident investigation |

# Thank You!

**Davis Vercher**

CS7346
Spring 2025