Cloud Computing
CSC547 - Fall 2023

Plagiarism Detection using Natural Language Processing

Mohammed Sami Ashfak MK (mamk)
Mushtaq Ahmed Shaikh (mshaikh2)

Plagiarism Declaration

We, the team members, understand that copying&pasting material from any source in our project is an allowed practice; we understand that not properly quoting the source constitutes plagiarism.

All team members attest that we have properly quoted the sources in every sentence/paragraph we have copy&pasted in our report. We further attest that we did not change words to make copy&pasted material appear as our work.

Contents

1 Introduction

1.1 Motivation

We selected this particular problem statement for our Cloud Computing Project with the aim of applying and refining the skills we've acquired throughout our lectures. Our goal is to implement a solution that integrates the various techniques and concepts learned over the course, drawing from the knowledge shared by our professors and the diverse literature we've consulted during our classes.

1.2 Executive Summary

With a steady increase in the number of sources of information, the need for a robust Software as a Service (SaaS) Plagiarism Detection service also increases. With multiple flaws like scalability issues, limited database coverage, and algorithmic constraints in the currently available Plagiarism Detection services businesses and individuals need a solution which is highly available, easily accessible, and better performing than the traditionally available solutions.
We wanted to design a solution that has a comprehensive and detailed plagiarism analysis unlike the traditionally available platforms. The users can submit text documents, essays or any other textual content for that matter to receive quick and accurate plagiarism reports. Being a Software as a Service (SaaS), our solution curbs the need for complex installations, optimizes costs and guarantees high-performance. This project recognizes the challenges of the current applications and instead provides a cloud-based, highly available solution that is designed according to the changing needs of the institutions and individuals.

2 Problem Description

2.1 The Problem

To design and develop a scalable and accessible Cloud-based Software as a Service (SaaS) Plagiarism Detection Solution that analyses texts ranging from essays to academic documents and provides a comprehensive plagiarism report which addresses the issues with the pre-existing solutions.

2.2 Business Requirement

The business requirements for the problem statement are as follows:

1. BR1 – Scalability: Ensure the system can seamlessly scale to handle growing content volumes through continuous resource monitoring.
2. BR2 – Availability: The plagiarism detection service should be available 24/7 with minimal downtime.
3. BR3 – Database Coverage: Maintain a comprehensive plagiarism detection database covering multiple sources.
4. BR4 – Cost Optimization: Utilize cloud resources efficiently to optimize costs.
5. BR5 – Security: Store user information and uploaded files securely.
6. BR6 – Performance: Provide low-latency access to the plagiarism detection service.
7. BR7 – Consistent High Performance: Consistently deliver high performance in terms of processing speed and responsiveness.
8. BR8 – Vendor Independence: Minimize dependencies on specific cloud service providers by avoiding vendor lock-in.
9. BR9 – Access: Cater to diverse user groups like students, professors, and research academia using access management techniques.
10. BR10 – Logging: Ensure comprehensive logging mechanisms to improve overall system performance.
11. BR11 – Reduce environmental impact by adopting sustainable practices in IT infrastructure and promote eco-friendly operations.

2.3 Technical Requirements

Following are the technical requirements for the problem statement:

1. TR1.1 – Implement dynamic resource allocation to meet increased customer demand. The capacity of resources such as CPU, memory should scale according to incoming workload. Maintain average server load of 75% and spawn new compute instances when load exceeds beyond 75%.
2. TR1.2 – Implement load balancing mechanisms to distribute incoming load across multiple servers/instances.
3. TR1.3 – Monitor system performance every 5 minutes, aiming for a 99% uptime and identifying performance bottlenecks within 1 minute.
4. TR1.4 – Implement caching strategies in the architecture to store frequently accessed data achieving cache hit ratio of 90%
5. TR2.1 – Maintain a system availability of 99%, ensuring no more than 5 minutes of downtime for auto recovery.
6. TR2.2 – Implement automatic failover mechanism ensuring failover occurs within 5 minute, minimizing service disruption.

7. TR3.1 – Develop a robust data integration framework which is capable of summing content from multiple sources.
8. TR3.2 – Choose a robust, scalable and high-performance database management system that can handle 10,000 requests per second.
9. TR3.3 – Take regular backups of the plagiarism detection database to create redundant copies of it.
10. TR4.1 – Implement tools for monitoring and analysing cloud resource usage and costs.
11. TR4.2 – Implement auto-scaling for resources on the basis of demand which allows the system to automatically scale up or down to optimize costs
12. TR4.3 – Implement automated scripts that help identify and remove the unused resources.
13. TR4.4 - Utilize cloud services that offer a pay-per-use pricing model to align costs with actual resource consumption.
14. TR5.1 – All data, including documents and plagiarism reports, must be encrypted during transmission and at rest to ensure confidentiality.
15. TR5.2 - Comply with relevant data protection regulations and industry standards, ensuring user data privacy.
16. TR5.3 - Employ secure key management practices and ensure data in transit and at rest is encrypted.
17. TR5.4 – Implement strict access control and authorization to limit access to user information and files.
18. TR5.5 – Implement robust disaster recovery mechanism so that the user information and files are secure.
19. TR6.1 – Implement geographical load balancing to redirect user request to the nearest data centre. We can deploy the application to multiple locations to meet reliability.
20. TR6.2 - Leverage in-memory databases and caching strategies to accelerate response times.
21. TR6.3 - Regularly monitor network latency in real time and implement optimisations based on observed performance.
22. TR6.4 - Utilize a real-time database that facilitates immediate data updates and synchronisation.
23. TR7.1 – Design the system architecture to scale horizontally, allowing it to handle increased load.
24. TR7.2 - Conduct regular load testing to simulate high usage scenarios and identify performance bottlenecks.
25. TR7.3 – Implement asynchronous processing to reduce user wait time and enhance overall user experience in handling time-intensive tasks such as running plagiarism check on a large file.
26. TR8.1 – Utilize cloud-agnostic technologies and frameworks to ensure compatibility across various cloud service providers.
27. TR8.2 - Implement containerization using technologies like Docker to abstract application dependencies and facilitate deployment on multiple cloud platforms.
28. TR8.3- Develop a comprehensive migration strategy that allows for easy transition between different cloud providers, minimizing the impact on the system's operation.
29. TR9.1 – Implement user segmentation mechanism to distinguish between various user groups like students, professors and research academia.

30. TR9.2 – Customize various workflows based on the specific requirements of each of the tenant groups like students, professors and research academia. For example, professors can directly notify students when plagiarism is detected.

31. TR9.3 – Implement Role Based Access Control (RBAC) to modulate permissions and access to relevant features for each of the tenant group.

32. TR10.1 - Implement a centralized logging system with the ability to capture, analyse, and store relevant logs for effective monitoring and troubleshooting.

33. TR10.2 - Configure log rotation and retention policies to manage log storage efficiently while ensuring critical information is retained for an appropriate duration.

34. TR11.1 - Select AWS instances which are designed for energy efficiency to reduce carbon footprint of the infrastructure.

35. TR11.2 - Use eco-friendly methods to recycle and dispose of electronic waste.

2.3.1 AWS Well Architected Framework

This section provides the list of Technical Requirements (TRs) grouped by the pillars of the AWS Well Architectured Framework.

Pillar	**Technical Requirement (TR)**
Operational Excellence	1.3, 3.1, 6.3, 6.4, 8.1, 8.2, 8.3, 9.2, 10.1, 10.2
Security	5.1, 5.2, 5.3, 5.4, 5.5, 9.1, 9.3
Reliability	2.1, 2.2, 3.2, 3.3
Performance Efficiency	1.1, 1.2, 1.4, 6.1, 6.2, 7.1, 7.2, 7.3
Cost Optimization	4.1, 4.2, 4.3, 4.4
Sustainability	11.1, 11.2

Table 1: Map of TR's with AWS WAF

2.4 Tradeoffs

1. **TR 1.2 conflicts with TR 6.1** Implementing load balancing mechanisms for scalability conflicts with the geographical load balancing requirement. Load balancing may direct traffic to servers that are not geographically closest to the user, potentially increasing latency.

2. **TR 1.4 conflicts with TR 3.1** Implementing caching strategies for improved performance conflicts with the need for a robust data integration framework. Caching may introduce challenges when integrating content from multiple sources, potentially leading to outdated or inconsistent data.

3. **TR 2.1 conflicts with TR 4.2** Redundant infrastructure for increased reliability conflicts with the goal of optimizing costs through auto-scaling. Maintaining redundant servers may incur additional costs, especially when scaling down during periods of low demand.

4. **TR 5.5 conflicts with TR 3.3** Implementing robust disaster recovery conflicts with the need for regular backups. While disaster recovery ensures data security, frequent backups may increase resource usage and potentially conflict with the disaster recovery plan.

5. **TR1.3 conflicts with TR7.2** Constant performance monitoring might conflict with asynchronous processing. Asynchronous tasks may not provide real-time data for monitoring tools, impacting their effectiveness.

6. **TR2.1 conflicts with TR7.3** Highly redundant infrastructure may conflict with asynchronous processing. Redundancy often involves synchronous processes for failover, while asynchronous processing aims to reduce user wait time.

7. **TR5.1 conflicts with TR8.2** Encrypting data during transmission and at rest may conflict with containerization. Containerization might require decryption at certain points, potentially introducing complexities in maintaining encryption.

8. **TR7.2 conflicts with TR10.1** Load testing may conflict with centralized logging. Intensive load testing could generate a significant volume of logs, potentially impacting the logging system's performance.

9. **TR6.2 conflicts with TR8.2** Leveraging in-memory databases might conflict with containerization. Containerization involves encapsulating applications and their dependencies, potentially affecting direct in-memory access.

10. **TR6.3 conflicts with TR4.1** Real-time network latency monitoring may conflict with tools for monitoring cloud resource usage and costs. Real-time monitoring could consume resources that might otherwise be used for cost analysis.

11. **TR8.1 conflicts with TR9.3** Cloud-agnostic technologies might conflict with Role-Based Access Control (RBAC). Some cloud-agnostic solutions may not fully align with specific RBAC implementations.

12. **TR5.3 conflicts with TR10.2** Implementing secure key management may conflict with log rotation policies. Strict key management might necessitate longer retention of cryptographic keys, impacting log rotation schedules.

13. **TR5.3 conflicts with TR7.1** Implementing secure key management conflicts with achieving high performance using horizontal scaling. Authentication and Authorization ensure security but degrades performance.

3 Provider Selection

3.1 Criteria for choosing a provider

The criteria for choosing a cloud provider are as follows:

- **Scalability:** The provider should have dynamic resource allocation and load balancing techniques to accommodate increasing volume of data while optimizing costs.
- **Reliability:** The provider should have minimum service interruptions and an automatic failover mechanism to ensure high reliability.
- **Cost Optimization:** The cloud provider should have mechanisms such as auto scaling and resource cleanup tools to minimize unnecessary usage of resources.
- **Cost Transparency:** The provider should have a clear and transparent pricing structure which makes it easier to estimate and manage costs effectively.
- **Support and SLAs:** It is necessary to consider the level of support and SLAs provided by the cloud provider to ensure timely resolution of issues.
- **Geographic Location:** The cloud provider should have its services available across the globe.
- **Comfort Level:** All team members should possess a high level of familiarity and comfort in working with the chosen cloud provider.

3.2 Provider Comparision

The following table provides a comparison of the cloud providers with respect to the above criteria. Each cloud provider is rated from 0 to 2, 2 being the highest score.

Provider	**AWS**		**Azure**		**IBM**	
	Justification	**Score**	**Justification**	**Score**	**Justification**	**Score**
Scalability	*AWS operates in in 81 availability zones across the globe	5/5	*Azure operates in 65 regions across the world.	4/5	*IBM has more than 60 datacenters in 19 countries across the globe.	3/5
Reliability	AWS[10] has had 4 service outages in the past 5 years starting from 2020	5/5	Azure's[17] outage history is 11 pages long for the past 5 years starting from 2020.	2/5	IBM[18] has had 10 outages for the past 5 years starting from 2020	3/5

Provider	AWS		Azure		IBM	
	Justification	**Score**	**Justification**	**Score**	**Justification**	**Score**
Cost Optimization	AWS[21] is the priciest when 67 cloud computing scenarios were compared by Flexera	3/5	Azure is cheaper than AWS but still pricier than IBM cloud	4/5	IBM is the cheapest cloud provider when 67 cloud computing scenarios were compared by Flexera	5/5
Support & SLA	AWS provides support during business-hours using email	4/5	Azure also provides support during business hours via email	4/5	IBM provides 24/7 support using a ticketing system with its free plan	5/5
Reputation	AWS[25] has the most market share compared to other cloud providers and hence has more reputation when compared to others (33.1% in US)	5/5	Azure[25] has the second largest market share in the US, around 20.9% and hence ranks second in terms of reputation in the comparison.	4/5	IBM[25] comes in the "Others" category of the market share and has the least market share when compared to its other two counterparts and hence has the least reputation.	5/5
Comfort Level	The team members have worked with AWS and are most familiar to it.	5/5	The team members have sparse information about services and products offered by Azure	3/5	The members in the team do not have any experience with the IBM cloud and its services.	2/5
Services Offered	AWS[24] offers more than 200 services for a wide range of technologies, industries and use cases.	5/5	Azure[23] provides more than 200 services that can be used to solve today's life challenges and create the future.	5/5	IBM[21] provides more than 170 services covering data, AI, containers, IoT, and blockchain.	4/5

*Note – Marked quantitative measures in the table have been updated as of January 2022.

3.3 Final Selection

The three cloud providers offer similar services. AWS has been around since 2006 and is a leading cloud provider with the most market share. It operates in 25 geographic regions with 81 availability zones and 218+ edge locations, ensuring high availability. AWS also offers the most services at reasonable prices. It's known for providing top-notch technical support in the market, addressing customer issues effectively. As developers, our past work experiences align well with AWS. Considering all the above criteria it has been decided to choose AWS cloud provider for this project.

3.4 Services offered by AWS

- **Amazon S3**[11] - Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance. It can store and protect any amount of data for several use cases, such websites, cloud-native applications, machine learning, and analytics. The objects of S3 are stored in the form of buckets. Amazon S3 claims to have aa 99.999999999% (11 9's) durability, and stores data for millions of customers all around the world.

- **Amazon Sagemaker**[20] - Amazon SageMaker is a fully managed machine learning service. It provides an integrated Jupyter authoring notebook instance for easy access to your data sources for exploration and analysis, so you don't have to manage servers. It also provides common machine learning algorithms that are optimized to run efficiently against extremely large data in a distributed environment

- **Amazon OpenSearch Service**[16] - OpenSearch is a distributed, community-driven, Apache 2.0-licensed, 100% open-source search and analytics suite used for a broad set of use cases like real-time application monitoring, log analytics, and website search. OpenSearch provides a highly scalable system for providing fast access and response to large volumes of data with an integrated visualization tool, OpenSearch Dashboards, that makes it easy for users to explore their data .

- **Amazon CloudWatch**[19] - Amazon CloudWatch monitors Amazon Web Services (AWS) resources and applications in real time. CloudWatch is used to collect and track variable metrics of resources and applications. Using CloudWatch one can either create alarms which are triggered when the metric crosses a threshold, or it can be be used to automatically make modifications to resources. For instance, we can monitor CPU usage and disk reads and writes of Amazon EC2 instances and then use this data to determine whether to launch additional instances to handle increased load or stop under-used instances to save cost.

- **AWS Lambda**[9] - It is a serverless, event-driven compute service used to run code for virtually any type of application or backend service without provisioning or managing servers. It is a pay as you use model where Lambda can be triggered from over 200 AWS services and software as a service (SaaS) application. The following are the brief use cases of AWS Lambda:

  1. Builds process workflows quickly and easily with suite of other serverless offerings and event triggers.
  2. Can be combined with AWS services to create secure, stable, and scalable online experiences such as interactive web and mobile backends.
  3. Enables powerful ML insights by pre-processing data before feeding it to your machine learning (ML) model.

- **AWS Identity and Access Management (IAM)**[8] - AWS IAM lets you define users with permissions across AWS resources, AWS Multi-Factor Authentication for privileged accounts, including options for software- and hardware-based authenticators. With IAM you can create groups and allow those users or groups to access some servers, or you can deny them access to the service

- **Amazon DynamoDB**[5] - Amazon DynamoDB is a serverless NoSQL database service that supports key-value and document data models with single-digit millisecond performance at any scale. It secures the data at rest, performs automatic backups and offers a guaranteed reliability with an SLA of up to 99.999% availability. Amazon DynamoDB uses AWS identity and access management (IAM) to authenticate, create, and access resources.

- **Amazon SQS**[13] - Amazon Simple Queue Service (Amazon SQS) is a fully managed message queuing service provided by Amazon Web Services (AWS). SQS enables decoupling of the components of a cloud application, allowing them to operate independently and scale independently. It reliably delivers large volumes of data with high throughput without losing messages. It also uses AWS key management techniques to securely transmit data between applications.

- **Amazon SNS**[12] - Amazon Simple Notification Service (Amazon SNS) is a fully managed messaging service provided by Amazon Web Services (AWS). It facilitates communication between different parts of the application or microservices using messages or notifications. The Amazon SNS sends notifications two ways, A2A and A2P. A2A provides high-throughput, push-based, many-to-many messaging between distributed systems, microservices, and event-driven serverless applications. A2P functionality sends messages to customers with SMS texts, push notifications, and email.

- **Amazon Comprehend**[4] - Amazon Comprehend is a natural language processing (NLP) service provided by Amazon Web Services (AWS). It makes it easy to extract insights and relationships from unstructured text by using machine learning techniques. Comprehend analyses text in multiple languages and supports a variety of use cases, including sentiment analysis, entity recognition, key phrase extraction, and language detection.

- **Amazon Textract** [14] - Amazon Textract is a machine learning (ML) service that automatically extracts text, handwriting, layout elements, and data from scanned documents. It goes beyond simple optical character recognition (OCR) to identify, understand, and extract specific data from documents. Textract uses ML to read and process any type of document, accurately extracting text, handwriting, tables, and other data with no manual effort.

- **Amazon Cognito**[3] - Amazon Cognito is a fully managed identity and access management service provided by Amazon Web Services (AWS). Using Amazon Cognito, you can add user sign-up and sign-in features and control access to your web and mobile applications.

- **Amazon EC2**[6] - Amazon Elastic Compute Cloud (Amazon EC2) is a web service provided by Amazon Web Services (AWS) that offers resizable compute capacity in the cloud. Users can easily launch virtual servers, known as instances, to host applications, websites, and other computing workloads. EC2 instances can be quickly scaled up or down to accommodate changing workloads, providing flexibility and cost efficiency. Users have full control over their instances, including the ability to start, stop, terminate, and monitor them.

- **Amazon EKS**[7] - Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that eliminates the need to install, operate, and maintain your own Kubernetes control plane on Amazon Web Services (AWS). Kubernetes is an open-source system that automates the management, scaling, and deployment of containerized applications.

- **Amazon VPC**[15] - Amazon Virtual Private Cloud (Amazon VPC) is a service provided by Amazon Web Services (AWS) that enables users to launch and operate a logically isolated section of the AWS Cloud. Within this virtual network, users can deploy AWS resources such as Amazon EC2 instances, Amazon RDS databases, and Amazon S3 buckets.

4 The first design draft

From the abovementioned technical requirements we choose the following for our first design draft:

1. TR1.1 – Implement dynamic resource allocation to meet increased customer demand. The capacity of resources such as CPU, memory should scale according to customer demand. Maintain average server load of 75% and spawn new compute instances when load exceeds beyond 75%. As the load drops below 25%, scale down the compute resources.
2. TR2.1 – Maintain a system availability of 99%, ensuring no more than 5 minutes of downtime for auto recovery.
3. TR9.1 – Implement user segmentation mechanism to distinguish between various user groups like students, professors and research academia.
4. TR4.1 – Use tools for monitoring and analysing cloud resource usage and costs.
5. TR7.3 – Implement asynchronous processing to reduce user wait time and enhance overall user experience in handling time-intensive tasks such as running plagiarism check on a large file.
6. TR4.2 – Implement auto-scaling for resources on the basis of demand which allows the system to automatically scale up or down to optimize costs.
7. TR4.4 - Utilize cloud services that offer a pay-per-use pricing model to align costs with actual resource consumption.
8. TR1.3 – Monitor system performance every 5 minutes, aiming for a 99% uptime and identifying performance bottlenecks within 1 minute.
9. TR6.1 – Implement geographical load balancing to redirect user request to the nearest data centre. We can deploy the application to multiple locations to meet reliability.
10. TR1.2 – Implement load balancing mechanisms to distribute incoming load across multiple servers/instances.
11. TR3.2 – Choose a robust, scalable and high-performance database management system that can handle up to 10,000 requests per second.
12. TR5.4 – Implement strict access control and authorization to limit access to user information and files.
13. TR5.1 – All data, including documents and plagiarism reports, must be encrypted during transmission and at rest to ensure confidentiality.
14. TR1.4 – Implement caching strategies in the architecture to store frequently accessed data achieving cache hit ratio of 90%.
15. TR7.1 – Design the system architecture to scale horizontally, allowing it to handle increased load.
16. TR6.2 - Leverage in-memory databases and caching strategies to accelerate response times.
17. TR6.3 - Regularly monitor network latency in real time and implement optimisations based on observed performance.
18. TR6.4 - Utilize a real-time database that facilitates immediate data updates and synchronisation.
19. TR8.2 - Implement containerization using technologies like Docker to abstract application dependencies and facilitate deployment on multiple cloud platforms
20. TR10.1 - Implement a centralized logging system with the ability to capture, analyse, and store relevant logs for effective monitoring and troubleshooting.
21. TR11.1 - Select AWS instances which are designed for energy efficiency to reduce carbon footprint of the infrastructure.

22. TR11.2 - Use eco-friendly methods to recycle and dispose of electronic waste.

We have constructed the block diagram to meet all specified requirements, as illustrated below:

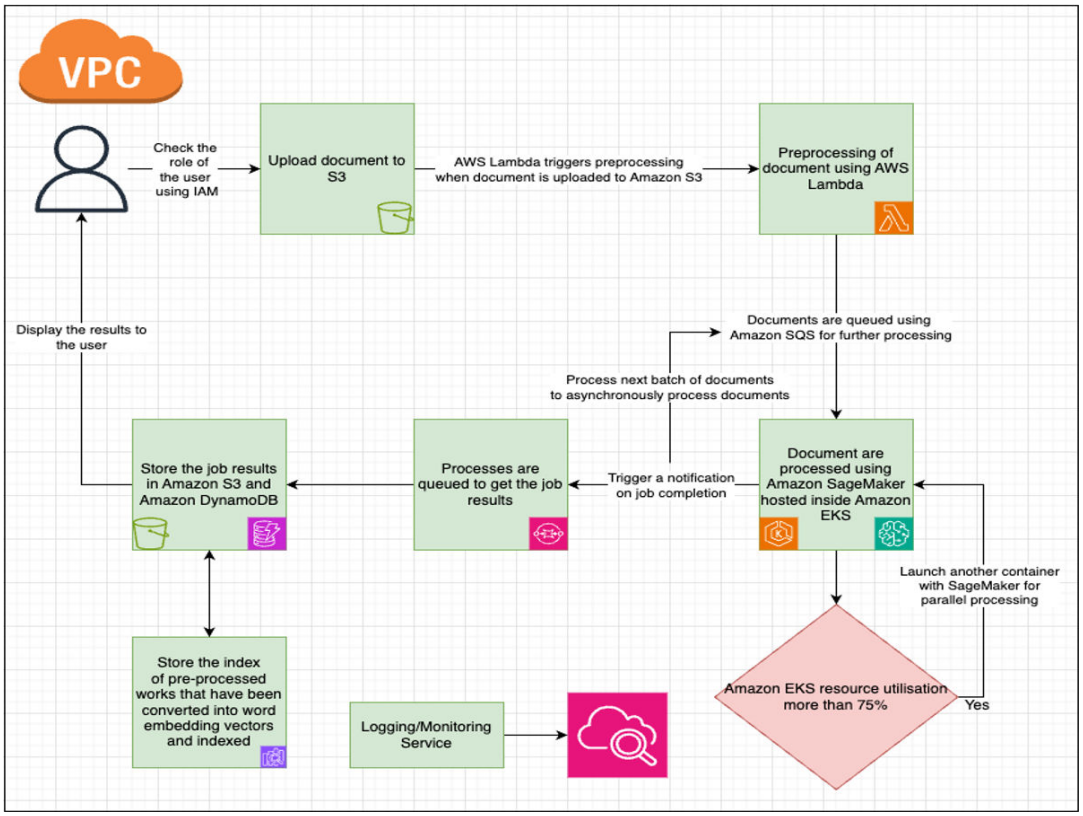


Figure 1: Block Diagram

4.1 The basic building blocks of design

The basic building blocks of the design are as follows:

- **Storage**
  - We have used Amazon S3 for storing the documents on which we have to run the Plagiarism Detection Solution.
  - Amazon OpenSearch Service is used for storing the indices of pre-processed words which will help in faster lookup of information during plagiarism detection.
  - We have used the Amazon DynamoDB to store the metadata of documents and user information.
- **Computing resources**
  - AWS Lambda is used as a serverless compute infrastructure that performs all the administrative tasks on the compute servers. The AWS Lambda function preprocesses documents and prepares document metadata and stores it in Amazon DynamoDB. It helps run the code without managing and provisioning any servers.
  - We have used Amazon EKS along with Amazon EC2 to dynamically deploy and run Kubernetes containers according to the load.
- **NLP Engine**
  - The NLP engine uses Amazon Textract for extracting text from uploaded documents and uses metadata to optimize extraction.

- We have used Amazon Comprehend which feeds on Amazon Textract's output and performs entity parsing, line/paragraph classification.
- The output from Comprehend is then used by the Machine Learning model deployed in production using Amazon SageMaker.

- **Networking**
  - We have used Amazon VPC to setup a private, isolated network within the AWS cloud.
  - Amazon SQS has been used to perform asynchronous processing of documents by queueing tasks.
  - Amazon SNS is used to trigger notification on task completion to SQS to move forward with the next batch of tasks.

- **Others**
  - We have used Amazon CloudWatch to monitor and analyze logs from all the used AWS services.
  - AWS IAM is used for Role Based Access Management (RBAC).
  - Amazon Cognito is used to control access of users to our web and mobile application

4.2 Top-level, informal validation of the design

- **Storage**
  - Amazon S3[11]: Amazon S3 is the preferred choice of storage for the files uploaded by the user. S3 provides simple a simple and secure object storage solution enabling users to store and retrieve their files. Since each object in S3 is stored in a bucket, it allows us to easily manage access permissions, versioning, and lifecycle policies to control data retention and archival. It is also highly available, durable and can be accessed with low-latency.
  - Amazon DynamoDB[5]: Amazon DynamoDB is a NoSQL database which stores the data in the form of key value pairs enabling quick access to data with low latency. The high read and write throughput of DynamoDB makes it the most optimal choice to store the document metadata and logs.
  - Amazon OpenSearch[16]: Amazon OpenSearch is designed for horizontal scalability and data visualization. It stores the index of the preprocessed files that have been converted into word embedding vectors and indexed. These stored indices can be used to fetch information about the already processed data.

- **Peformance**
  - AWS Lambda[9]: AWS Lambda is a serverless computing service that helps us run our program without maintaining or managing the servers. We have used AWS Lambda to run functions to preprocess text data when an event is triggered.
  - Amazon EKS[7]: Amazon EKS is a fully managed Kubernetes service provided by Amazon. It can be used to scale up or scale down when the load on a container in EKS varies thereby providing elastic computation.
  - Amazon SageMaker[20]: Amazon SageMaker is a fully managed Machine Learning service provided by AWS. It is the best choice for us to build, train and deploy Machine Learning Model directly into the production environment. We achieve high scalability by running SageMaker inside an EKS container which can be replicated when the resource utilization exceeds a pre-defined threshold (in our case it is 75%).

- **Networking**
  - Amazon SQS[13]: Amazon SQS is a fully managed message queuing service. It is an optimal solution to facilitate asynchronous processing of large text files.

- Amazon SNS[12]: Amazon SNS is a fully managed messaging service provided by Amazon. SNS is used to trigger notification on task completion to SQS to process the next batch of tasks that it has queued.

- **Security**
  - Amazon S3[11]: Amazon S3 provided various levels of encryption to data in transit and rest. It provides Server Side Encryption, Client Side Encryption, Encryption in Transit and Bucket Policies and ACL.
  - Amazon DynamoDB[5]: DynamoDB integrates with AWS Identity and Access Management (IAM) for authentication and authorization. We can define fine-grained access controls to DynamoDB by integrating it with Identity and Access Management (IAM).
  - Identity and Access Management[8]: IAM is used to securely control access to AWS resources. It helps in managing various groups of users and allows the definition of policies to specify permissions for accessing resources.

4.3 Action items and rough timeline

Skipped

5 The second design

5.1 Use of Well-Architected framework[2]

The steps mentioned by the Well Architected framework are as follows:

1. Curate a list of all the Business Requirements (BR) for which we need to produce an architecture.
2. For each of these Business Requirements (BR) try writing Technical Requirements(TR).
3. Search for various options to address and solve the Technical Requirements.
4. Evaluate tradeoffs for each of the identified services in step 3.
5. After proper evaluation and comparison between each of the options select an option to move with.
6. Document the reasons for the selection in a design document and draw architectural diagram

5.2 Discussion of Pillars[2]

As discussed in section 4.1, the AWS Well Architected Framework (WAF) guides cloud architects to design a highly reliable, scalable and sustainable cloud infrastructure on Amazon Web Services for their applications. In this section, we describe the 4 pillars of WAF- Operational Excellence, Reliability, Security, and Performance Efficiency. By following these principles and best practices defined in WAF, we aim to utilize the full potential of cloud services for our architecture.

1. **Operational Excellence:** The Operational Excellence pillar plays an integral part in implementing best practices for the design, delivery, and maintenance of AWS workloads. It ensures that organizations align their strategies with business objectives, effectively run workloads, gain operational insights, and continually enhance supporting processes for delivering business value. The AWS WAF provides 5 design principles of operational excellence in cloud.
   (a) **Perform Operations as Code**
      - Apply the same engineering discipline to the entire cloud environment as applied to application code.
      - Define the entire workload, including infrastructure and applications, as code.
      - Script and automate operations execution to minimize human error and ensure consistency.

   (b) **Make Frequent, Small, Reversible Changes**
      - We need to continuously improve the operating procedures as the workload changes.
      - Conduct regular reviews of the procedures to ensure that all the teams are familiar with them.

   (c) **Refine Operations Procedures Frequently**
      - We need to continuously improve the operating procedures as the workload changes.
      - Conduct regular reviews of the procedures to ensure that all the teams are familiar with them.

   (d) **Anticipate Failure**
      - Regularly simulate failure events to test workloads and check for the team's response.
      - Identify potential failure sources through pre-emptive testing measures.
      - Continuously test the response procedures for the test failure scenarios.

   (e) **Learn from All Operational Failures**
      - Drive improvements in applications and infrastructure based on the experiences from operational events.
      - Build a culture of sharing information across the entire organization.

The Operational Excellence in AWS workloads can be achieved by having best practices in these four areas.

(a) **Organization** To achieve Operational Excellence, teams must share a common understanding of the workload, their roles, and business objectives. Prioritizing efforts based on customer needs, governance, and compliance ensures alignment with organizational goals. Teams need to regularly review and update priorities to adapt to changing needs. Additionally, we need to evaluate business threats, risks, and trade-offs to ensure the right decision making.

(b) **Prepare** Preparing for operational excellence involves designing workloads for observation through telemetry, including metrics, logs, and events. Teams need to adopt approaches for fast feedback, bug fixing, which accelerates changes into production. Teams also need to evaluate operational readiness, using runbooks and playbooks along with assessing risks.

(c) **Operate** In the operational phase, operational success can be achieved by defining expected outcomes and metrics for workload. Efficient management of planned and unplanned operational events is critical, utilizing runbooks for known events and playbooks for investigations. Teams need to prioritize responses based on business and customer impact to ensure effective incident resolution. There should be clear communication of operational status through dashboards and notifications to keep all the stakeholders informed.

(d) **Evolve** Continual evolution is key to operational excellence, involving dedicated cycles for continuous incremental improvements. Performing post-incident analysis, identifying contributing factors, and implementing preventative actions enhance resilience. Teams need to regularly evaluate and prioritize improvement opportunities for both workload and operations to ensure optimization. Teams need to incorporate continuous improvement through feedback loops and sharing lessons learned.

2. **Security:** The Security pillar explains the ability to encrypt and protect data, systems and assets to take advantage of cloud technologies to make it more secure. There are 7 design principles to maintain security in the cloud:

   (a) **Implement a strong identity foundation:** Adhere to the principle of granting the minimum necessary privileges and enforce the division of responsibilities by ensuring proper authorization for every interaction involving your AWS resources.

   (b) **Enable traceability:** Monitor, notify, and audit actions in your environment in real-time. Integrate monitoring services to automatically investigate and take action against threats.

   (c) **Apply security at all layers:** Utilize a layered defense strategy incorporating multiple security measures. Implement this approach across various levels.

   (d) **Automate security best practices:** Automated security mechanisms make it easily to scale up rapidly and cost-effectively.

   (e) **Protect data in transit and at rest:** Use encryption, tokenization and access controls on data both in transit and at rest.

   (f) **Keep people away from data:** Utilize automated tools to prevent manual handling of data. This reduces the risk of human induced errors while handling sensitive data

   (g) **Prepare for security events:** Be ready for potential incidents by establishing incident management and investigation policies and processes that align with your organizational needs.

   There are six best practice areas for security in the cloud. They are as follows:

   (a) **Security:** To ensure the secure operation of your workload, it is essential to apply comprehensive best practices across all security aspects. This involves extending the requirements and processes established for operational excellence at both the organizational and workload levels to all areas. Staying informed about AWS and industry recommendations, along with threat intelligence, aids in the continuous evolution of your threat model and control objectives. The automation of security processes, along with testing and validation, enables the scalability of your security operations.

(b) **Identity and Access Management:** It plays a crucial role in security as it makes sure that only users with proper authentication and authorization can access the resources. IAM is the primary service for privilege management, allowing control over user and programmatic access to AWS services. IAM facilitates secure access for systems requiring AWS access through roles, instance profiles, identity federation, and temporary credentials.

(c) **Detection:** Detective frameworks are important in identifying potential security threats and incidents. Different types of Detective controls help in effective decision making and establishment of operational baselines. Implementing detective controls involves processing logs, events, and continuous monitoring.

(d) **Infrastructure protection:** Infrastructure protection involves various control methods needed to fulfill the best practices. The use of these methods is important for successful ongoing operations in either the cloud or on-premises.

(e) **Data protection:** Prior to designing a system, it is crucial to establish foundational security practices. Data classification enables the categorization of organizational data based on sensitivity levels, while encryption safeguards data by making it unintelligible to unauthorized access. These practices are essential for achieving objectives such as preventing financial loss and ensuring compliance with regulatory obligations.

(f) **Incident response:** Despite having advanced preventive and detective controls, organizations should establish processes for responding to and mitigating the impact of security incidents. The architecture of your workload significantly influences your team's effectiveness in responding to incidents, isolating systems, and restoring operations to a known good state. Preparing tools and access in advance, coupled with regular incident response practice through simulations or "game days," ensures that your architecture can support timely investigation and recovery during security incidents.

5.3 Use of CloudFormation diagram

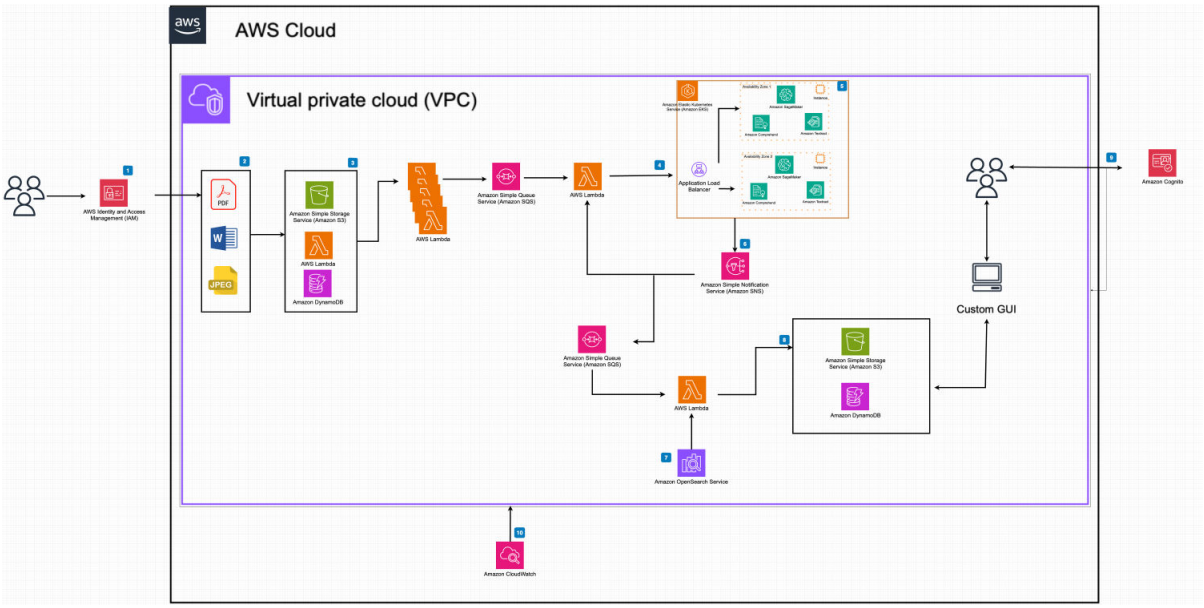


Figure 2: Cloud Formation Diagram

5.4 Validation of the Design

In this section we justify our design to meet the selected technical requirements in section 4.

1. The NLP Engine which consists of Amazon SageMaker, Amazon Comprehend, Amazon Textract is hosted inside an Amazon EC2 instance where we run the NLP Engine as containers inside

Amazon EKS. The worker nodes of Amazon EKS scale up and down based on the load. We use the Horizontal Pod Autoscaler which creates new pods when the CPU utilization exceeds 75%. The pods automatically scale down when the CPU utilization decreases less than 25%. The formula to create new replicas is as shown below:

$$\text{new\_replica} = \lceil \text{current\_replicas} \times \frac{\text{current\_metric\_value}}{\text{desired\_metric\_value}} \rceil$$

2. We refer the Amazon Well Architected Framework to achieve a goal of 99% availability to meet TR2.1.

   - We use Amazon S3 bucket to store the files uploaded by the user. This ensures that we always achieve 99.9% availability of the service.
   - We deploy our applications in multiple regions and availability zones for fault-tolerance and resiliency.
   - We use auto-scaling using the Horizontal Auto Pod Scaler to satisfy time varying workloads.
   - We use Amazon DynamoDB to store metadata of the files. Amazon DynamoDB performs frequent data backups and helps always achieve 99.999% availability.
   - We use AWS Lambda which is a serverless compute service that helps us run the preprocessing tasks without physically managing the resources. AWS Lambda has inbuilt functionality to run the function in multiple availability zones and triggers services on demand. This helps ensure that the AWS Lambda function is available to process files in case of service interruption in a single zone.
   - Using the AWS WAF, we calculate 99% availability of our application as follows:

     $$\text{availability} = \frac{\text{availability for use time}}{\text{total time}}$$

     Considering a year with 365 days, an availability if 99% means that the service would be down for 3 days 15 hours if we have one failure every month and it takes about 60 minutes to recover from a failure. In addition, we also have 20 failures which can recover automatically taking 5 minutes per failure.

     $$\text{Total downtime} = 12 \times (1 \text{ hour}) + 20 \times (5 \text{ minutes}) = 13 \text{ hours } 40 \text{ minutes}$$

     We can also estimate the availability using the formula as shown below:

     $$\text{Available}_{\text{est}} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

     Here, the Mean Time Between Failure (MTBF) is 30 days (720 hours), and Mean Time To Recover (MTTR) is 1 hour.

     $$\text{Available}_{\text{est}} = \frac{720}{720 + 1} = 0.99861 \approx 99.9\% \text{ availability.}$$

3. We use AWS Identity Management, User Authentication and Authorization (AWS IAM) along with Amazon Cognito to identify tenants and isolate features shared between tenants. These services help us optimize the tenant requirements. In our case, the tenants are academia which includes students, professors and researchers. These services also help us manage and provide fine grained access controls for workloads.

4. We use Amazon CloudWatch to monitor and troubleshoot infrastructure. We monitor key metrics and logs, visualize application and infrastructures, create alarms and correlate data to understand and resolve the root cause of performance issues in our architecture. The alarms facilitated the alarms facilitated by Amazon CloudWatch helps you to watch the metric values against the thresholds and detect anomalous behaviour when an alarm is triggered the Amazon EC2 instances are either auto scaled up or scaled down.

5. Implement asynchronous processing for to process large files simultaneously this in turn reduces the user with time and enhances the overall experiences of user while handling time intensive tasks such as running plagiarism on large files. We use Amazon SQS and Amazon SNS services to facilitate a synchronous processing of large files. Amazon SQS queues tasks and provides them to the Amazon SageMaker in an ordered format and Amazon SNS triggers a notification when the task is completed by Amazon SageMaker and informs Amazon SQS to send in the next process in the queue.

6. We have used multiple cost effective AWS resources to achieve optimum cost for our architecture. First we use AWS Lambda, which is a serverless compute service, which automatically scales up and down based on the load. This ensures that we achieve a pay per use model for the AWS Lambda service. Second we use Amazon CloudWatch to monitor the resources in the architecture. Amazon CloudWatch has thresholds built in and once the load exceeds beyond the threshold the resources are automatically scaled up. This ensures that we are not wasting resources and help us achieve optimum cost.

7. To restrict the resource usage and to limit the cost of the architecture we have used pay per use services throughout the architecture. For instance we use AWS Elastic Kubernetes Services to horizontally scale the pod based on the load. With the increase in the load the resources increases and the resources are scaled down when the demand of this services goes down. This way we can customize and change the configuration according to our needs. The horizontal auto pod scalar thus help us achieve cost surges.

8. We monitor the system performance every 5 minutes aiming to achieve 99% uptime and try to identify the performance bottlenecks within one minute. For instance, Amazon DynamoDB internally creates replicas of the metadata of the files. This helps us ensure that we have the metadata replicas available at all times. In addition the Amazon S3 storage service guarantees an availability of 11.9 seconds per year. This ensures that the service that is used to store the files is available at all times thereby helping us achieve 99% uptime.

9. We utilise geographic load balancing to redirect the user request to nearest data center. We also deploy the application 2 multiple availability zones to meet reliability for instance Amazon S3, Amazon Dynamo DB are deployed across several availability zones. The AWS Lambda function inherently runs in multiple availability zones. Additionally the Amazon EC 2 instances are spanned across multiple availability zones to provide greater availability and reliability. The EC2 automatically balances the number of instances per availability zone.

10. We implement load balancing mechanisms to distribute incoming load across multiple servers and instances.

11. We choose a robust commerce scalable and high performance database system that can handle up to 5,500 requests per second. There are 2 storage services used in our architecture. First, Amazon S3 which is an object storage service used to securely store files.

    - Amazon S3 can handle 5,500[1] requests per second to retrieve data and 3,500 requests per second to add data.
    - DynamoDB can achieve up to 20 million[22] requests per second depending upon the number of RCU's and WCU's used.

12. We implement strict access control and authorization using Amazon IAM. Additionally, we also use Amazon Cognito to provide fine grained permissions to users at web/mobile application levels. The level of access can be easily configured using these services and prevents unauthorized groups from accessing user data.

13. All data including documents, files, and plagiarism reports are encrypted during transmission and at rest to ensure confidentiality. Amazon S3 and DynamoDB have built-in encryption. The network traffic within the Virtual Private Cloud is end-to-end encrypted.

14. We have Amazon OpenSearch to store the indices of pre-processed works that have been converted into word embedding vectors and indexed. This helps us implement caching strategies in the architecture to store pre-processed data achieving the hit ratio of 90%.

5.5 Design principles and best practices used.[2]

The best practices that we have implemented are as follows:

- **Organization (WAF Operational Excellence):** We prioritize efforts based on customer needs and conduct regular review meetings to update priorities to adapt to changing needs.
- **Evolve (WAF Operational Excellence):** We use Amazon CloudWatch to monitor and log all the incidents that occur within the architecture. We perform post incident analysis to identify factors contributing to failure and implement preventive actions to enhance resiliency.
- **Identity and Access Management (WAF Security):** We use Amazon IAM service which allows us to control users access to AWS services and resources. We apply granular policies to assign permissions to our user group that includes academia (students, professors and universities).
- **Data Protection (WAF Security):** The AWS services such as Amazon S3 and DynamoDB are used to encrypt user data and file metadata and manage keys including regular key rotation. The network data is also encrypted within the Virtual Private Cloud.
- **Foundation (WAF Reliability):** We manage the service quotas and constraints using Amazon CloudWatch. We have also set the threshold of 75% to increase the resources dynamically.
- **Change Management (WAF Reliability):** We monitor the resource utilization using Amazon CloudWatch and add logs to Amazon DynamoDB. These logs and metrics provide powerful insights to the health of our workload. We have configured the workload in such a way that notifications are sent when thresholds are crossed or failures occur. In case of failures, it can recover automatically in response. We also have scalable workload to provide elasticity by adding or removing resources automatically to meet the current demand.
- **Failure Management (WAF Reliability):** Amazon DynamoDB creates frequent backups of the data that can be used in case of disruptions to the data. Additionally, services like Amazon S3 help us achieve low MTTR (Mean Time To Recovery). We also have backups in place as a measure for Disaster Recovery (DR).

The Design Principles we have used are as follows:

- **Go global in minutes (WAF Performance Efficiency):** We deploy the workloads in multiple AWS regions around the world to provide low latency and better experience for our customers at minimal cost.
- **Use serverless architecture (WAF Performance Efficiency):** Serverless architecture removes the need to run and maintain physical servers. For instance, we have used AWS Lambda which is a serverless service that helps you deploy your application without managing the physical resources.
- **Scale horizontally to increase aggregate workload availability (WAF Reliability):** We have deployed the NLP Engine inside Amazon EKS which is deployed inside the Amazon EC2. This helps us replace one large resource with multiple smaller resources to reduce the impact of single point of failure.
- **Protect data in transit and at rest (WAF Security):** Amazon S3 and Amazon DynamoDB encrypt the data during transmission and at rest which helps secure the data.

5.6 Tradeoffs revisited

1. [**Redundant infrastructure conflicts with cost optimization using auto-scaling**] This tradeoff revolves around striking a delicate balance between making sure that the system is resilient and reducing the operational expenses. Using a redundant infrastructure with additional backup servers increases the reliability and fault tolerance of the system, reducing the downtime by mitigating impact of failures. On the other hand, auto-scaling dynamically adjusts

the resources based on demand for the software by scaling up the resources during periods of heavy demand and scaling down during non-peak hours. The conflict is introduced while trying to maintain an equilibrium between maintaining redundancy and cost-optimization. In our case we chose redundant infrastructure because the documents and files uploaded by the users and the plagiarism reports generated by the software are of utmost importance when compared to cost-optimization using auto-scaling.

2. [**Robust disaster recovery conflicts with the need for regular backups**]
   Disaster Recover (DR) and regular backups are two important aspects of the IT Infrastructure. The primary goal of disaster recovery is to ensure business continuity and data security in the event of a disaster. This is achieved by setting up redundant systems and data backups in geographically separate locations. On the other hand, regular backups are essential for data integrity and quick recovery from data corruption and non-catastrophic incidents. The tradeoff arises from the fact that both disaster recovery and regular backups are resource-intensive processes, and there's often a conflict in how frequently these processes should be executed. Allocating more resources to one aspect (e.g., frequent backups) might limit the resources available for the other (e.g., disaster recovery), and vice versa. Assuming that the probability of a catastrophic event is significantly less compared to design and application failures, we prioritize allocating more resources to regular backups and reducing the frequency of Disaster Recovery backups.

3. [**Real-time network latency monitoring may conflict with tools for monitoring cloud resource usage and costs**]
   The tradeoff conflicts between two aspects of IT infrastructure management. Real-time network latency monitoring is crucial for ensuring optimal performance and responsiveness of applications and services. Tools are employed to monitor network latency and address issues quickly. On the other hand, Monitoring cloud resource usage and costs is essential for optimizing resource allocation, managing expenses, and ensuring cost-effectiveness in a cloud computing environment. Both technical requirements need dedicated resources. Devoting resources to real-time network latency monitoring might limit the availability of resources for activities like cost analysis and optimization. Additionally, the real-time nature of network latency monitoring might lead to increased costs. As a result, we have decided to choose managing cloud resources and cost over monitoring network latency. We want to use resources as efficiently as possible and reduce the overall cost of the design.

4. [**Constant performance monitoring conflicts with asynchronous processing**]
   The conflict between constant performance monitoring and asynchronous processing lies in the need for real-time data for monitoring and the inherent characteristics of asynchronous processes. Constant performance is the process of tracking and logging various metrics like response times, resource utilization, and error rates to analyze and detect irregularities in working. On the other hand, asynchronous processes run independently of the main program, allowing it to process multiple files simultaneously increasing the performance and making it easier to scale. Asynchronous processing might introduce a lag in data availability which stops the continuous flow of data to the performance monitoring service and in turn introduces a lag in identifying performance-related issues. We have prioritized asynchronous processing in our project as we might need to process very huge documents which we plan to process asynchronously. If we prioritize constant performance monitoring the system will take time to process large documents which will eventually decrease the performance of the system.

5. [**Implementing secure key management conflicts with achieving high performance using horizontal scaling**]
   The conflict highlights the challenge in IT security and performance optimization. Secure key management is crucial for protecting sensitive information such as keys for encryption and decryption. Horizontal scaling involves adding more hardware or resources to a system to improve its performance and handle increased load. The conflict arises when secure key management introduces additional processing overhead, impacting the system's ability to horizontally scale for high performance. Horizontal scaling aims to distribute the load across multiple nodes, improving overall system performance. However, the additional security measures may introduce

delays, potentially decreasing the benefits of auto scaling. Considering these tradeoffs, we decided to proceed with high performance. One of the most important goals of our design is to achieve high performance. This can be achieved by horizontal scaling and hence we decided to prioritize it by having our NLP engine run within the Amazon EKS clusters.

6. [**Leveraging in-memory databases conflicts with containerization**]
The conflict between leveraging in-memory databases and containerization stems from the advantages of in-memory databases and the encapsulation principles of containerization. In-memory databases store and manage the system's RAM and not the traditional disk storage. This helps in faster retrieval and data access. This improves the performance of the data-intensive applications. On the other hand, containerization involves encapsulating an application with its data, runtime environment, and dependencies into an isolated unit known as container. The conflict arises due to the fact that in-memory databases rely on direct access to the applications memory for faster lookup of data whereas containerization encapsulates the underlying hardware and creates an isolated memory. In our project we have chosen containerization over in-memory databases because we have deployed NLP Engines inside these containers and we might need to duplicate these containers when the demand is high. If we use in-memory databases it will increase the processing times during peak hours and eventually decrease the performance.

5.7 Discussion of an alternate design

Skipped

6 Kubernetes experimentation

6.1 Experiment Design

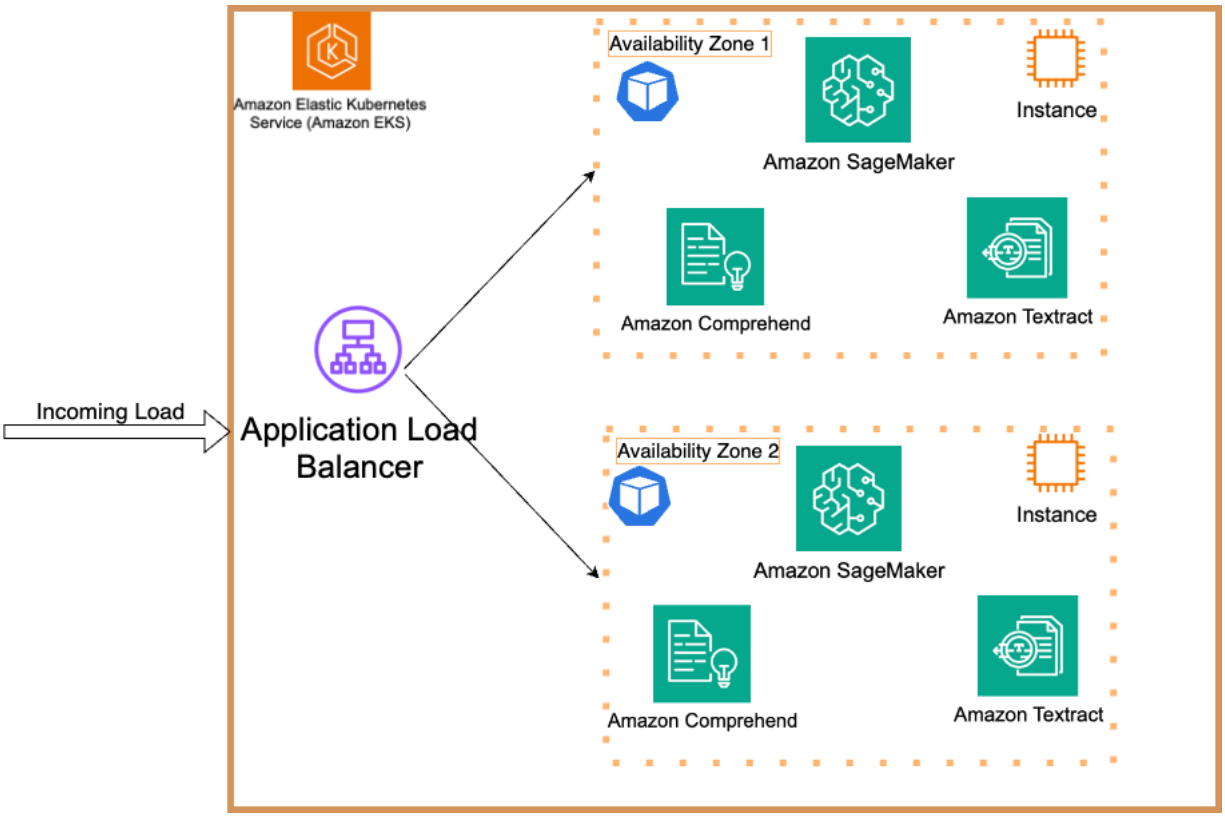


Figure 3: Experiment Design

- Experiment Details:

  1. In this experiment we are going to test one of the major components of our Plagiarism Detection Service, i.e., the NLP Engine. The NLP Engine is used to extract text from the documents and determine whether the given text is plagiarized.

  2. Workflow of NLP Engine:

     - When a client uploads a document to Amazon S3, a few pre-processing steps are performed before the request is sent to the NLP Engine.
     - The Application Load Balancer receives the request and forwards it to the containers deployed by Amazon EKS.
     - Each of these containers has the NLP Engine hosted in it. The Application Load Balancer forwards the request to the containers according to the current load.
     - The NLP Engine inside the container performs the processing on the document and forwards the result to the Amazon SQS (Simple Queue Service) for further processing.

  3. Need of Kubernetes in our solution:

     - **Scaling**: Plagiarism Detection solutions can have varying workloads, especially in an academic setting, during peak submission times. Using Kubernetes helps in easily scaling up the application by creating multiple containers with the NLP Engine to handle the increased workload.

- **Resource Utilization:** Kubernetes helps in efficiently using the resources by automatically scaling up during peak times and scaling down when there is not a lot of workload. This is important for our application as it might have to process large datasets.
- **Fault Tolerance:** Kubernetes improves the fault tolerance due to the features like automatic container restarts and health checks. If a node fails, Kubernetes will automatically direct the load to a healthy node.
- **Isolation and Security:** Plagiarism Detection system often deals with sensitive data. Kubernetes isolates the containers and secures the communication between them which helps maintain security and intgrity.

4. Use of Kubernetes in our experimentation:
   - Kubernetes provides us the opportunity to automatically scale the system depending on the incoming workload.
   - In our solution we have used Amazon CloudWatch to continuously monitor the resource utilization and performance.
   - We have deployed the NLP Engine inside the Kubernetes container.
   - We need to set threshold values to decide when to create a new container and when to terminate one. We have set the threshold to create a new container as 75% and the threshold to terminate a container as 25%.
   - When the resource utilization on a container exceeds 75% a new container is created with the NLP Engine in it.
   - Similarly, when the resource utilization of a container decreases less than 25%, the container is terminated as the workload is less enough that there is no need of this container.
5. Handling of loads in the service:
   - We have used 1 Kubernetes node containing 2 pods initially.
   - As per our criteria there can be minimum of 1 pod and maximum of 10 pods.
   - We have set the threshold for automatic scaling as 75% of the mean CPU utilization.
   - If the mean CPU utilization exceeds 75% then a new pod will be spawned by the autoscaler to mange the increased load.
   - We have set the threshold to terminate a pod as 25% of the mean CPU utilization.
   - When the mean CPU utilization reduces below 25% a pod is automatically terminated as it is not needed due to the low load.
   - We have used locust tool to generate large amounts of load to test our Kubernetes autoscaler.

- Services used during the experiment:

1. **Automatic Load Balancer:** An automatic load balancer is used to distribute the load across multple EC2 instances which contain Amazon EKS in which the NLP Engine is hosted.
2. **Amazon Elastic Kubernetes Service:** Amazon EKS is a Kubernetes service provided by Amazon which simplifies deployment, management and scaling of containerized applications using Kubernetes. We have hosted our NLP Engine inside the EKS containers.
3. **Amazon EC2:** Elastic Compute Cloud is a web service provided by Amazon which provides re-sizable compute capabilities. It does this using virtual servers which are known as instances. EC2 helps scale the resources up or down based on demand.

4. **Kubernetes Node:** Kubernetes Node is a physical or virtual machine that runs the containerized applications. Nodes are worker machines that form clusters execute the tasks assigned to them.

5. **Kubernetes Pods:** A Kubernetes Pod is the smallest deployable unit. A pod contains one or more containers that share the same storage and network. Pods serve as the basic building blocks in deploying an application using Kubernetes.

6. **Kubernetes Container:** A Kubernetes container is a standalone and lightweight software package that is sufficient to run a piece of software. Containers help provide consistency across environments and make it easier to deploy applications.

7. **Kubernetes HPA**: Horizontal Pod Autoscaler is used to automatically adjust the number of pods depending on the mean CPU utilization. HPA helps in auto-scaling of resources to handle the time-varying workloads.

6.2 Workload generation with Locust

- Locust is an Open Source load testing tool that is used to measure the performance and scalability of applications. Locust is used to simulate a large number of users trying to access the application which lets us know how the application behaves under heavy workload.
- In our project we have used locust to test how efficiently our system scales up and down when the load varies with time.
- Here are the configurations we have used to test with Locust:

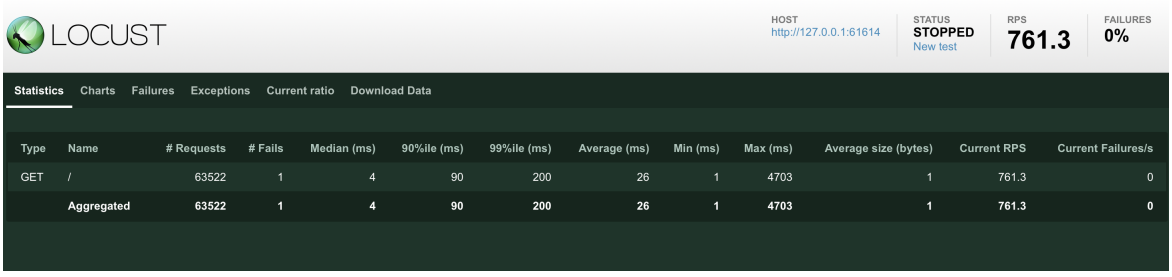
    1. Number of users during peak concurrency - 40
    2. Spawn rate - 0.5
    3. Total number of requests - 63522
    4. Requests per second - 793.17
    5. Failures per second - 0.01

- Steps to run tests using Locust:

    - Run the locust.py file which runs the Locust server.
    - Initialize the values for peak concurrency and spawn rate and provide the link to the target host.
    - Run the test

- The YAML files can be found at this repository - https://github.com/mksami22/CSC547-MK-Shaikh

Figure 4: Locust waveforms



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	63522	1	4	90	200	26	1	4703	1	761.3	0
	Aggregated	**63522**	**1**	**4**	**90**	**200**	**26**	**1**	**4703**	**1**	**761.3**	**0**

Figure 5: Locust logs

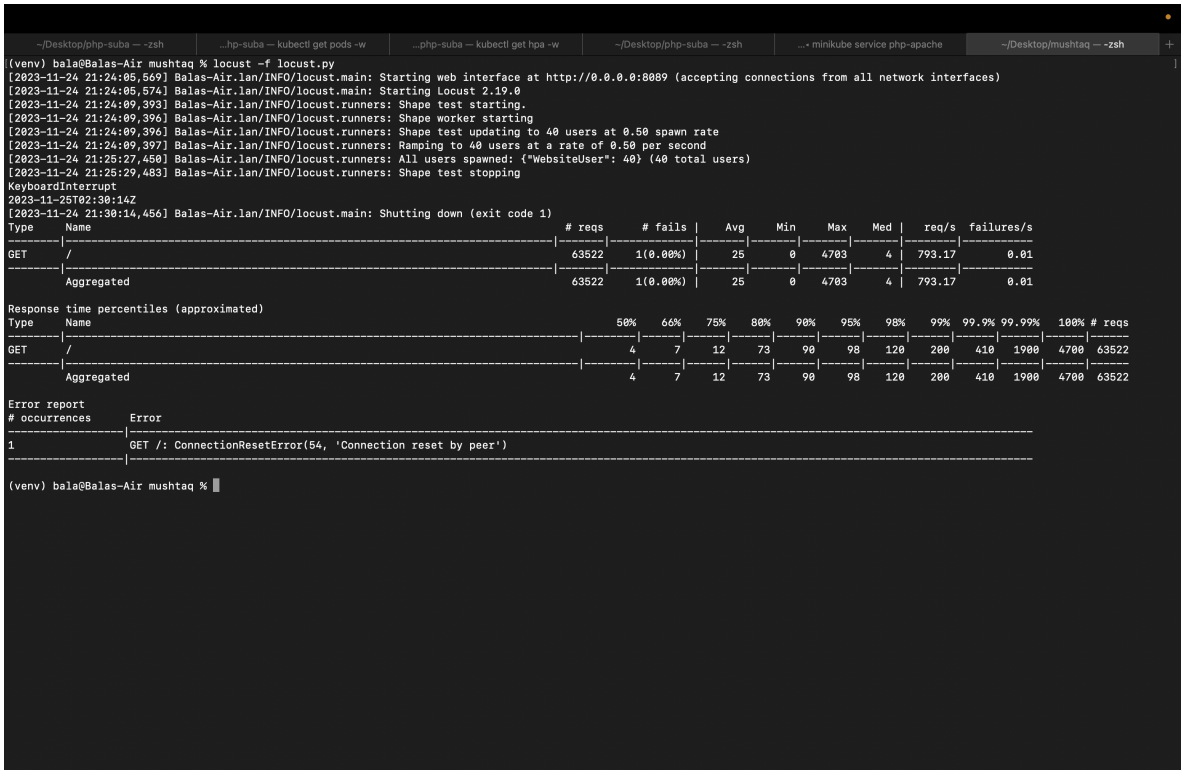


Figure 6: Locust terminal

6.3 Analysis of the results

1. Initially we have set the minimum pods to 1 and maximum number of pods to 10. The number of replicas at the beginning is 1.

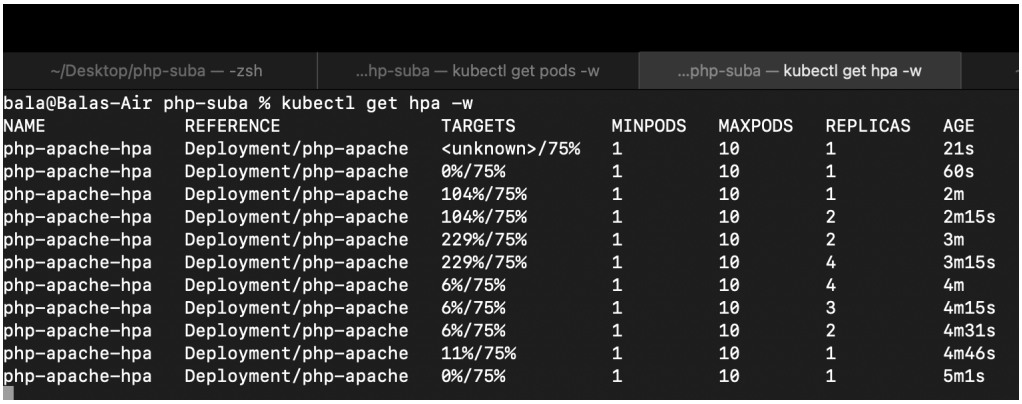


Figure 7: HPA Scaler

2. We then added a custom HPA scaler that scales the number of replicas when the average CPU load increases to 75%. Further, we have used the "behavior" parameter to simulate the Scale-Down scenario.

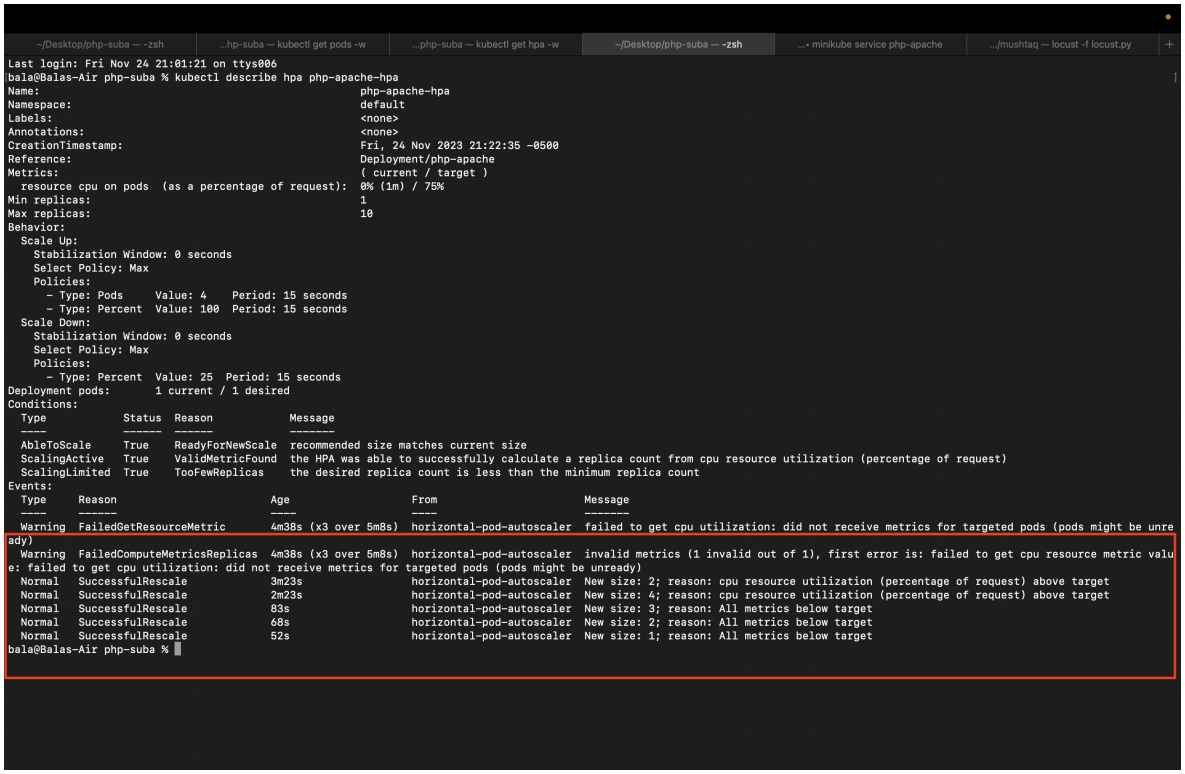


Figure 8: HPA Describer

3. The behavior for ScaleDown polls for the target CPU utilisation every 15 seconds. As soon as it observes the CPU utilisation go below 25%, it immediately starts to scale down the replicas as shown in Figure 8. We have set the "stabilizationWindowSeconds" parameter to 0 to see immediate change.

4. Analyzing the results shown in Figure 7, initially the number of replicas are 1. As the CPU utilisation increases to 229% the number of replicas increase to 4 which is expected as per the scale up formula.

5. After few seconds, as the CPU utilisation decreases to 6%, the replicas start to scale down from 4 to 1. Since, the CurrentMetric is 6% which is well within the TargetMetric of 25%, we see the replicas scale down immediately.

6. When the load testing ends, the CPU utilisation goes to 0%. In this case we do not need any extra replicas. As a result, the hpa sclaer sets the number of replicas to its default value which is 1.

Test Scenarios	**Expected Output**	**Actual Output**
When the load is high (Avg CPU utilization¿75%)	Number of replicas should scale till the Avg CPU utilization decreases to 60%	We see that when the CPU utilization exceeds 75% and reaches to 229% the number of replicas increased to 4. This is as expected.
When the load is low (Avg CPU utilization¡25%)	The number of replicas should scale down and keep doing so until the replicas=1	We see that when the CPU utilization reaches 6% which is within the target threshold of 25%, the replicas immidiately start to scale down from 4 to 1. This is the expected output.

Table 2: Test Results

7 Ansible playbooks

Skipped

8 Demonstration

Skipped

9 Comparisons

Skipped

10 Conclusion

10.1 Lessons Learned

The key learning from the project are as follows:

- Constructing the Business Requirements and deriving the Technical Requirements consumed a significant amount of time, but this effort facilitated the construction of all other sections, making the overall project development more manageable.
- This project proved to be a valuable opportunity to apply key learnings to a practical problem statement, allowing us to reinforce and solidify concepts learned throughout the lectures from day one.
- The project introduced us to the use of CloudFormation diagrams, providing insights into designing a cloud architecture for a real-world problem using various AWS services.
- Our knowledge about the diverse range of AWS services and their practical applications was significantly expanded through the execution of this project.
- Hands-on experience with Kubernetes and Horizontal Pod Scaler enhanced our understanding of how applications scale in a real-world context, offering valuable insights into the dynamics of application scalability.

10.2 Possible continuation of the project

We intend to pursue the development of a functional prototype based on the identified problem statement as part of a personal project. We currently have no plans to enroll in the Advanced Cloud course during the upcoming Spring Semester.

11 References

References

[1] https://aws.amazon.com/about-aws/whats-new/2018/07/amazon-s3-announces-increased-request-rate performance/.

[2] https://aws.amazon.com/architecture/well-architected/?wa-lens-whitepapers.sort-by=item.additionalFields.sortDatewa-lens-whitepapers.sort-order=descwa-guidance-whitepapers.sort-by=item.additionalFields.sortDatewa-guidance-whitepapers.sort order=desc.

[3] https://aws.amazon.com/cognito/.

[4] https://aws.amazon.com/comprehend/.

[5] https://aws.amazon.com/dynamodb/.

[6] https://aws.amazon.com/ec2/.

[7] https://aws.amazon.com/eks/.

[8] https://aws.amazon.com/iam/.

[9] https://aws.amazon.com/lambda/.

[10] https://aws.amazon.com/premiumsupport/technology/pes/.

[11] https://aws.amazon.com/s3/.

[12] https://aws.amazon.com/sns/.

[13] https://aws.amazon.com/sqs/.

[14] https://aws.amazon.com/textract/.

[15] https://aws.amazon.com/vpc/.

[16] https://aws.amazon.com/what is/opensearch/.

[17] https://azure.status.microsoft/en us/status/history/.

[18] https://cloud.ibm.com/status/incident reports.

[19] https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html.

[20] https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html.

[21] https://rb.gy/94dnzx.

[22] https://rb.gy/c4clwc.

[23] https://rb.gy/o2dl0i.

[24] https://rb.gy/ru2owi.

[25] https://www.linkedin.com/pulse/cloud-market-share-whos-winning-losing-sanjay-t s/.