Laboratory Assignment 1: Deployment View of Mathpix

Can Acay

March 28, 2024

Contents

1 Software System Selection **3**

2 Quality Attributes and System Requirements **3**
2.1 Performance . . . 3
2.2 Scalability . . . 3
2.3 Reliability . . . 3
2.4 Security . . . 3

3 Component Identification **3**

4 Deployment Models and Nodes **4**

5 Deployment Patterns and Styles Analysis **5**
5.1 Client-Server Pattern . . . 5
5.2 Microservices Architecture . . . 5
5.3 API Gateway Pattern . . . 5
5.4 Database per Service Pattern . . . 5
5.5 Event-Driven Architecture . . . 5

6 Deployment Tactics Evaluation **5**
6.1 Horizontal Scaling . . . 5
6.2 Caching . . . 6
6.3 Monitoring and Logging . . . 6

7 Report **6**
7.1 Analysis Summary . . . 6
7.2 Rationale for Deployment Patterns, Styles, and Tactics . . . 6
7.3 Alignment with System Requirements and Quality Attributes . . . 7

8 Appendix: Deployment Diagram **8**

9 Diagram Description **8**
9.1 Client Application . . . 8
9.2 Authenticator Service . . . 9
9.3 API Gateway . . . 9
9.4 Load Balancer . . . 9
9.5 OCR Service . . . 9
9.6 LaTeX Rendering . . . 9
9.7 User Database . . . 9
9.8 Monitoring & Analytics Server . . . 9
9.9 Metrics & Logs Database . . . 10
9.10 Analytics Database . . . 10
9.11 Input Database . . . 10

1 Software System Selection

Mathpix is an OCR (Optical Character Recognition) service designed to extract textual information from images. Its purpose is to accurately convert images of typed or handwritten math problems into editable text and render LaTeX code for mathematical expressions. Key functionalities include image preprocessing, text recognition, LaTeX rendering, and handling user interactions through various client applications.

2 Quality Attributes and System Requirements

The Mathpix system should be designed to meet the following key quality attributes and system requirements:

2.1 Performance

- The OCR service should process images and generate LaTeX code within an acceptable response time, typically under 2 seconds for average-sized images.
- The system should be able to handle a high volume of concurrent requests without significant performance degradation.
- Caching mechanisms should be implemented to improve response times for frequently accessed results.

2.2 Scalability

- The system should be designed to scale horizontally, allowing the addition of more instances to handle increased traffic and processing demands.
- The microservices architecture should enable independent scaling of individual components based on their specific resource requirements.
- The system should be able to accommodate a growing user base and increasing usage without compromising performance or availability.

2.3 Reliability

- The system should have high availability, with a minimum uptime of 99.9
- Fault tolerance mechanisms should be implemented to handle component failures gracefully, without impacting the overall system functionality.
- Regular backups and disaster recovery procedures should be in place to minimize data loss and enable quick recovery in case of failures.

2.4 Security

- User authentication and authorization should be enforced to protect user data and prevent unauthorized access.
- Secure communication protocols (HTTPS) should be used for all data transmissions between components and client applications.
- Sensitive user information and processed data should be encrypted at rest and in transit to maintain confidentiality.
- Regular security audits and vulnerability assessments should be conducted to identify and mitigate potential security risks.

3 Component Identification

Mathpix is built with a range of components that work together to provide a robust service:

- **Client Applications**: Interfaces include a web application, a mobile app, and a desktop program, each designed to meet different user needs.

- **Input Handlers**: Tools to handle various types of inputs, including keyboard for text, images for scanning, handwriting recognition, and PDF text extraction.
- **Authenticator Service**: Ensures users can securely access their accounts.
- **API Gateway**: Serves as the hub for all client requests, efficiently managing traffic and optimizing system response.
- **Rate Limiter and Response Cache**: Helps maintain system stability by controlling the flow of requests and caching data to speed up responses.
- **Monitoring & Analytics Server**: Keeps an eye on system health and gathers data to help improve the service.
- **Databases**: Stores various types of data, including user information, input records, performance metrics, and cached LaTeX renderings.
- **OCR Service**: Recognizes text in images and converts it into an editable format.
- **LaTeX Rendering**: Takes the text from the OCR and turns it into LaTeX format, ready for use in academic and scientific documents.

4 Deployment Models and Nodes

1. **Client Application (Public Cloud or User Devices):** The client applications, such as web, mobile, or desktop apps, should be deployed on public cloud platforms or run directly on user devices. This allows easy access for users to interact with the Mathpix service.
2. **API Gateway (Public Cloud):** The API Gateway should be deployed on a public cloud to handle incoming requests from client applications. It acts as the entry point for the system, routing requests to the appropriate services.
3. **Load Balancer (Public Cloud):** The load balancer should be deployed on a public cloud to distribute incoming traffic across multiple instances of the API Gateway, OCR Service, and LaTeX Rendering components.
4. **Authenticator Service (Private Cloud or Dedicated Server):** The Authenticator Service should be deployed on a private cloud or dedicated server to ensure the security of user authentication and authorization processes. Keeping it separate from public-facing components adds an extra layer of security.
5. **OCR Service (Private Cloud or Dedicated Server):** The OCR Service, which performs the core functionality of image processing and text recognition, should be deployed on a private cloud or dedicated server. This ensures better control over the processing resources and allows for scalability and performance optimization.
6. **Input Database (Private Cloud or Dedicated Server):** The Input Database, which stores the processed images and extracted text, should be deployed on a private cloud or dedicated server. This keeps the data secure and allows for efficient access by the OCR Service.
7. **Monitoring & Analytics Server (Private Cloud or Dedicated Server):** The Monitoring & Analytics Server should be deployed on a private cloud or dedicated server to collect and process system metrics and logs. This allows for centralized monitoring and analysis of the system's performance and usage.
8. **Metrics & Logs Database (Private Cloud or Dedicated Server):** The Metrics & Logs Database should be deployed on a private cloud or dedicated server to store the collected metrics and logs. This ensures data security and allows for efficient querying and analysis.
9. **Analytics Database (Private Cloud or Dedicated Server):** The Analytics Database, which stores user behavior and system usage data, should be deployed on a private cloud or dedicated server. This keeps the data secure and allows for efficient access by the Analytics Server.
10. **LaTeX Rendering (Private Cloud or Dedicated Server):** The LaTeX Rendering component should be deployed on a private cloud or dedicated server to handle the rendering of mathematical expressions into LaTeX code. This ensures better control over the rendering process and allows for scalability.

11. **User Database (Private Cloud or Dedicated Server):** The User Database, which stores user account information, should be deployed on a private cloud or dedicated server to ensure the security and privacy of user data.

12. **Image Processing (Private Cloud or Dedicated Server):** The Image Processing component, which performs tasks like noise reduction, binarization, and normalization, should be deployed on a private cloud or dedicated server. This allows for efficient processing and scalability of the image preprocessing tasks.

5 Deployment Patterns and Styles Analysis

Based on the identified components and the chosen deployment models and nodes, the following deployment patterns and styles can be applied to the Mathpix system:

5.1 Client-Server Pattern

The Mathpix system can follow a client-server pattern, where the client applications (web, mobile, and desktop) interact with the server-side components through the API Gateway. The client applications send requests to the server, which processes them and returns the appropriate responses. This pattern allows for a clear separation of concerns between the client and server, enabling scalability and maintainability.

5.2 Microservices Architecture

The Mathpix system can be designed using a microservices architecture, where the system is decomposed into smaller, loosely coupled services. Each component, such as the OCR Service, LaTeX Rendering, Image Processing, and various databases, can be developed and deployed as independent microservices. This architecture promotes modularity, flexibility, and scalability, as each microservice can be scaled and updated independently based on the specific requirements.

5.3 API Gateway Pattern

The API Gateway acts as the entry point for all client requests and serves as a single interface to the underlying microservices. It handles request routing, composition, and protocol translation. The API Gateway can also implement cross-cutting concerns such as authentication, rate limiting, and caching. This pattern simplifies client communication and provides a unified API for the Mathpix system.

5.4 Database per Service Pattern

Each microservice in the Mathpix system can have its own dedicated database, following the Database per Service pattern. This allows each service to have its own data model and schema, optimized for its specific requirements. The User Database, Input Database, Metrics & Logs Database, and Analytics Database can be deployed separately and accessed by their respective services, ensuring data isolation and scalability.

5.5 Event-Driven Architecture

The Mathpix system can incorporate an event-driven architecture to enable asynchronous communication between microservices. Events can be used to trigger actions and data flow between services. For example, when an image is processed by the OCR Service, it can publish an event containing the extracted text. The LaTeX Rendering service can subscribe to this event and render the LaTeX code asynchronously. This pattern promotes loose coupling and enables scalability and fault tolerance.

6 Deployment Tactics Evaluation

To achieve specific quality attributes in the Mathpix system, the following deployment tactics can be evaluated:

6.1 Horizontal Scaling

Horizontal scaling can be applied to improve the performance and scalability of the Mathpix system. The API Gateway, OCR Service, and LaTeX Rendering components can be deployed across multiple instances, with a load balancer distributing the incoming requests among them. This allows the system to handle increased traffic and ensures high availability. Kubernetes or other container orchestration platforms can be used to manage the deployment and scaling of these components.

6.2 Caching

Caching can be implemented at various levels to improve the performance and reduce the load on the backend services. The API Gateway can include a response cache to store frequently accessed results, such as rendered LaTeX code or recognized text. This reduces the need to process the same requests repeatedly. Additionally, the LaTeX Rendering service can maintain a cache of rendered expressions to serve them quickly for subsequent requests. Caching can significantly improve response times and reduce the overall system load.

6.3 Monitoring and Logging

Comprehensive monitoring and logging should be implemented to ensure the reliability and observability of the Mathpix system. The Monitoring & Analytics Server can collect metrics and logs from various components and store them in the Metrics & Logs Database. This allows for real-time monitoring of system health, performance, and usage. Tools like Prometheus and Grafana can be used for collecting and visualizing metrics, while the ELK stack (Elasticsearch, Logstash, Kibana) can be employed for centralized logging and analysis. Alerts can be set up based on predefined thresholds to proactively identify and resolve issues.

7 Report

7.1 Analysis Summary

The deployment of the Mathpix system has been carefully designed to ensure high performance, scalability, reliability, and security. The system follows a microservices architecture pattern, which allows for the decomposition of the system into smaller, loosely coupled services. This approach promotes modularity, flexibility, and maintainability, as each microservice can be developed, deployed, and scaled independently based on its specific requirements.

The client applications, such as the web, mobile, and desktop apps, interact with the system through the API Gateway, which serves as the single entry point for all client requests. The API Gateway acts as a reverse proxy, routing requests to the appropriate microservices based on the request path or parameters. It also handles cross-cutting concerns such as authentication, rate limiting, and caching, simplifying client communication and providing a unified API for the Mathpix system.

To ensure high availability and scalability, the API Gateway, OCR Service, and LaTeX Rendering components are deployed across multiple instances using a load balancer. The load balancer distributes incoming traffic evenly among the instances, allowing the system to handle increased loads and ensuring optimal performance. Horizontal scaling is employed to dynamically adjust the number of instances based on the demand, enabling the system to accommodate a growing user base and increasing usage without compromising performance or availability.

The deployment diagram showcases a clear separation between the public-facing components (client applications and API Gateway) and the internal microservices and databases. The public-facing components are deployed on a public cloud infrastructure, providing easy access for users and enabling seamless interaction with the Mathpix service. On the other hand, the core microservices and databases, such as the OCR Service, LaTeX Rendering, Image Processing, and various databases, are deployed on a private cloud or dedicated servers. This deployment approach enhances security and control over sensitive data and critical processing tasks.

To further improve performance and reduce the load on backend services, caching mechanisms are implemented at various levels. The API Gateway includes a response cache to store frequently accessed results, such as rendered LaTeX code or recognized text. This eliminates the need to process the same requests repeatedly, significantly reducing response times. Additionally, the LaTeX Rendering service maintains a cache of rendered expressions, allowing for quick retrieval of previously rendered content.

To ensure the reliability and observability of the Mathpix system, comprehensive monitoring and logging mechanisms are implemented. The Monitoring & Analytics Server collects metrics and logs from various components and stores them in the Metrics & Logs Database. This enables real-time monitoring of system health, performance, and usage. Tools like Prometheus and Grafana are employed for collecting and visualizing metrics, while the ELK stack (Elasticsearch, Logstash, Kibana) is used for centralized logging and analysis. Alerts are set up based on predefined thresholds to proactively identify and resolve any issues, minimizing downtime and ensuring a smooth user experience.

7.2 Rationale for Deployment Patterns, Styles, and Tactics

The deployment patterns, styles, and tactics chosen for the Mathpix system are based on a careful consideration of the system requirements and desired quality attributes.

The microservices architecture pattern is selected to achieve modularity, scalability, and maintainability. By breaking down the system into smaller, loosely coupled services, each component can be developed, deployed, and scaled independently. This allows for faster development cycles, easier updates, and the ability to scale individual components based on their specific resource requirements.

The client-server pattern is employed to establish a clear separation between the client applications and the server-side components. This separation allows for better security, as the server-side components can be deployed on a private cloud or dedicated servers, protecting sensitive data and critical processing tasks.

The API Gateway pattern is chosen to provide a single entry point for all client requests and to handle cross-cutting concerns. It simplifies client communication, enables request routing, and allows for the implementation of common functionalities such as authentication, rate limiting, and caching.

The database per service pattern is selected to ensure data isolation and allow each microservice to have its own optimized data model and schema. This approach prevents data coupling between services and enables independent scaling and evolution of each service's data storage.

Horizontal scaling and load balancing are employed to achieve high performance and availability. By distributing the load across multiple instances of the API Gateway, OCR Service, and LaTeX Rendering components, the system can handle increased traffic and ensure optimal response times. Horizontal scaling allows for dynamic adjustment of resources based on demand, ensuring the system can accommodate a growing user base and increasing usage.

Caching is implemented to improve performance and reduce the load on backend services. By caching frequently accessed results, such as rendered LaTeX code or recognized text, the system can serve requests faster and minimize unnecessary processing.

Monitoring and logging are crucial for ensuring the reliability and observability of the system. By collecting metrics and logs from various components and storing them in a centralized database, the system can be proactively monitored for any issues or anomalies. Real-time monitoring and alerting enable quick identification and resolution of problems, minimizing downtime and ensuring a smooth user experience.

7.3 Alignment with System Requirements and Quality Attributes

The deployment choices made for the Mathpix system align well with the identified system requirements and desired quality attributes.

Performance is achieved through horizontal scaling, load balancing, and caching. By distributing the load across multiple instances and caching frequently accessed results, the system can handle a high volume of concurrent requests and deliver fast response times.

Scalability is addressed through the microservices architecture and horizontal scaling. The system can easily scale individual components based on their specific resource requirements, allowing it to accommodate a growing user base and increasing usage without compromising performance or availability.

Reliability is ensured through the separation of concerns between public-facing and internal components, as well as comprehensive monitoring and logging. By deploying critical services on a private cloud or dedicated servers and implementing real-time monitoring and alerting, the system can proactively identify and resolve issues, minimizing downtime and ensuring a reliable user experience.

Security is enhanced by deploying the Authenticator Service on a separate, secure server and using secure communication protocols and data encryption. The separation of public-facing and internal components further protects sensitive data and critical processing tasks.

In conclusion, the deployment patterns, styles, and tactics chosen for the Mathpix system align well with the system requirements and desired quality attributes. The microservices architecture, client-server pattern, API Gateway, database per service pattern, horizontal scaling, caching, monitoring, and logging all contribute to achieving high performance, scalability, reliability, and security. By carefully designing the deployment architecture and selecting appropriate patterns and tactics, the Mathpix system is well-positioned to deliver a robust and efficient OCR service to its users.

8 Appendix: Deployment Diagram

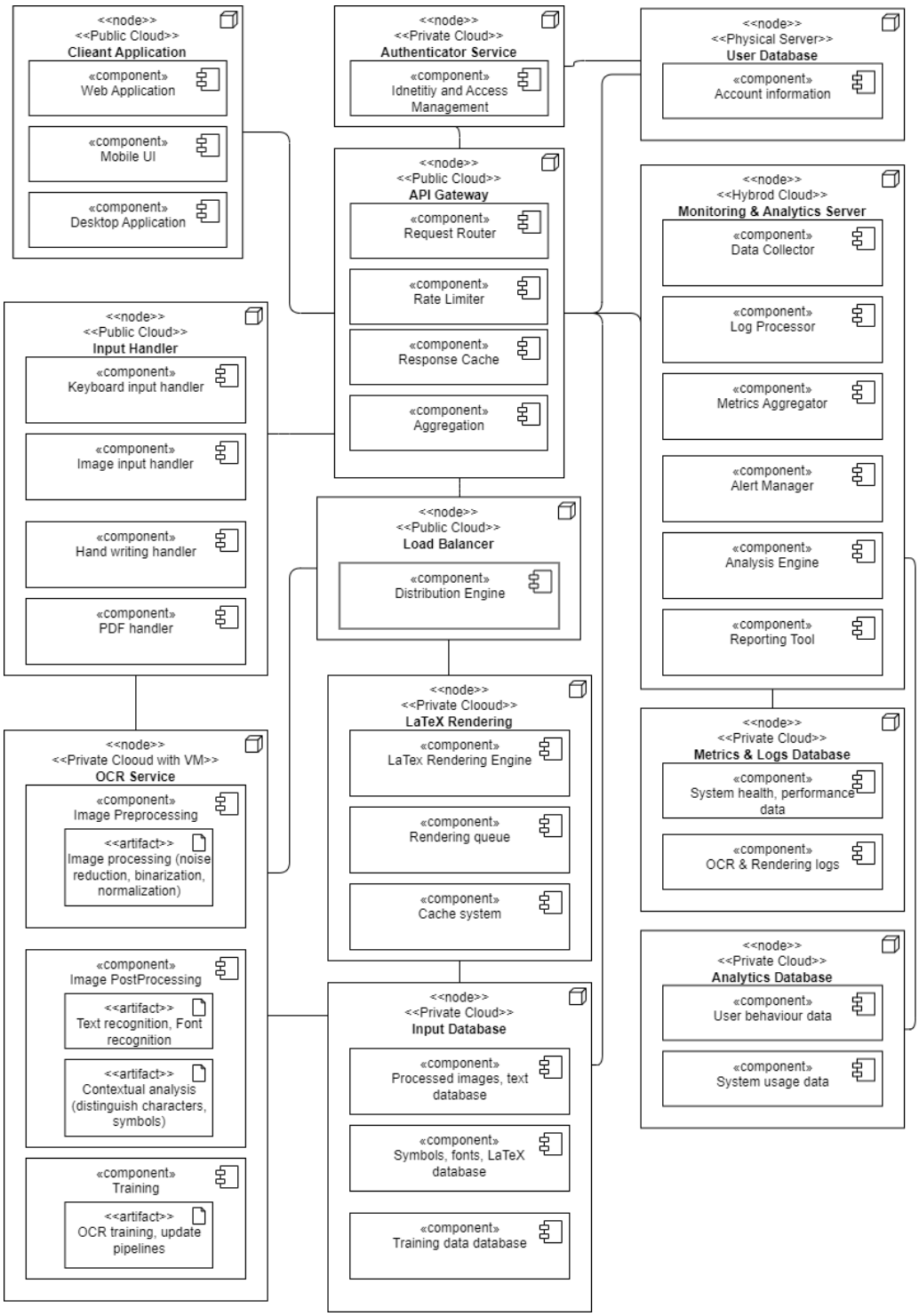

Figure 1: Deployment Diagram of Mathpix

9 Diagram Description

9.1 Client Application

Components:

- Web Application: A browser-based interface for users to access OCR services.
- Mobile UI: A mobile user interface designed for accessing services on smartphones.

- Desktop Application: A standalone application for desktop computers.

9.2 Authenticator Service

Components:

- Identity and Access Management: Manages user credentials and access rights.

9.3 API Gateway

Components:

- Request Router: Directs incoming API requests to appropriate services.
- Rate Limiter: Prevents overuse of resources by limiting request rates.
- Response Cache: Temporarily stores responses to improve performance.
- Aggregation: Combines data from multiple services for a single response.

9.4 Load Balancer

Components:

- Distribution Engine: Distributes incoming traffic across servers efficiently.

9.5 OCR Service

Components:

- Image Preprocessing: Enhances images for better OCR results.
- Image PostProcessing: Refines OCR results for accuracy.
- Training: Improves OCR accuracy by machine learning from new data.

Artifacts:

- OCR Training and Update Pipelines: Processes for updating the OCR models.

9.6 LaTeX Rendering

Components:

- LaTeX Rendering Engine: Converts text to LaTeX format.
- Rendering Queue: Manages the order in which rendering tasks are processed.
- Cache System: Stores frequently accessed data to improve performance.

9.7 User Database

Components:

- Account Information: Stores user profiles and authentication data.

9.8 Monitoring & Analytics Server

Components:

- Data Collector: Gathers data on system use and performance.
- Log Processor: Analyzes log files to extract useful information.
- Metrics Aggregator: Compiles various metrics into a unified view.
- Alert Manager: Manages alerts for system events and issues.
- Analysis Engine: Performs deep analysis of collected data.
- Reporting Tool: Generates reports on analytics and metrics.

9.9 Metrics & Logs Database

Components:

- System Health and Performance Data: Contains metrics on system performance.
- OCR & Rendering Logs: Stores logs specific to OCR and LaTeX rendering tasks.

9.10 Analytics Database

Components:

- User Behaviour Data: Information on how users interact with the system.
- System Usage Data: Records of how the system resources are used.

9.11 Input Database

Components:

- Processed Images and Text Database: Stores images after OCR processing and the extracted text.
- Symbols, Fonts, LaTeX Database: Repository of symbols and fonts used for LaTeX rendering.
- Training Data Database: Contains data used to train the OCR system.