**PROJECT REPORT**

**Image to PDF converter**

Anshul Choudhary (E22CSEU0952)

Mohit Jakhar(E22CSEU0941)

Akshit Nagaar(E22CSEU0944)

A report submitted in part fulfilment of the degree of

**B.Tech in Computer Science**

**Supervisor:** Dr. Naween Kumar

****

**School of Computer Science Engineering and Technology**

**Bennett University**

**April 29,2025**

1. **Introduction**

This project presents a serverless web application that allows users to upload multiple images and convert them into a single PDF document. It leverages a set of AWS cloud services to create a scalable, secure, and fully managed solution. The application demonstrates the effective use of serverless architecture for handling real-world workloads with minimal infrastructure overhead.

1. **Problem Statement**

Creating PDFs from images is a common requirement for many users (e.g., students, professionals). Desktop software solutions may not be available or easy to use. This project provides an easy-to-access, browser-based alternative powered entirely by cloud services without maintaining any backend server infrastructure.

1. **Objectives**

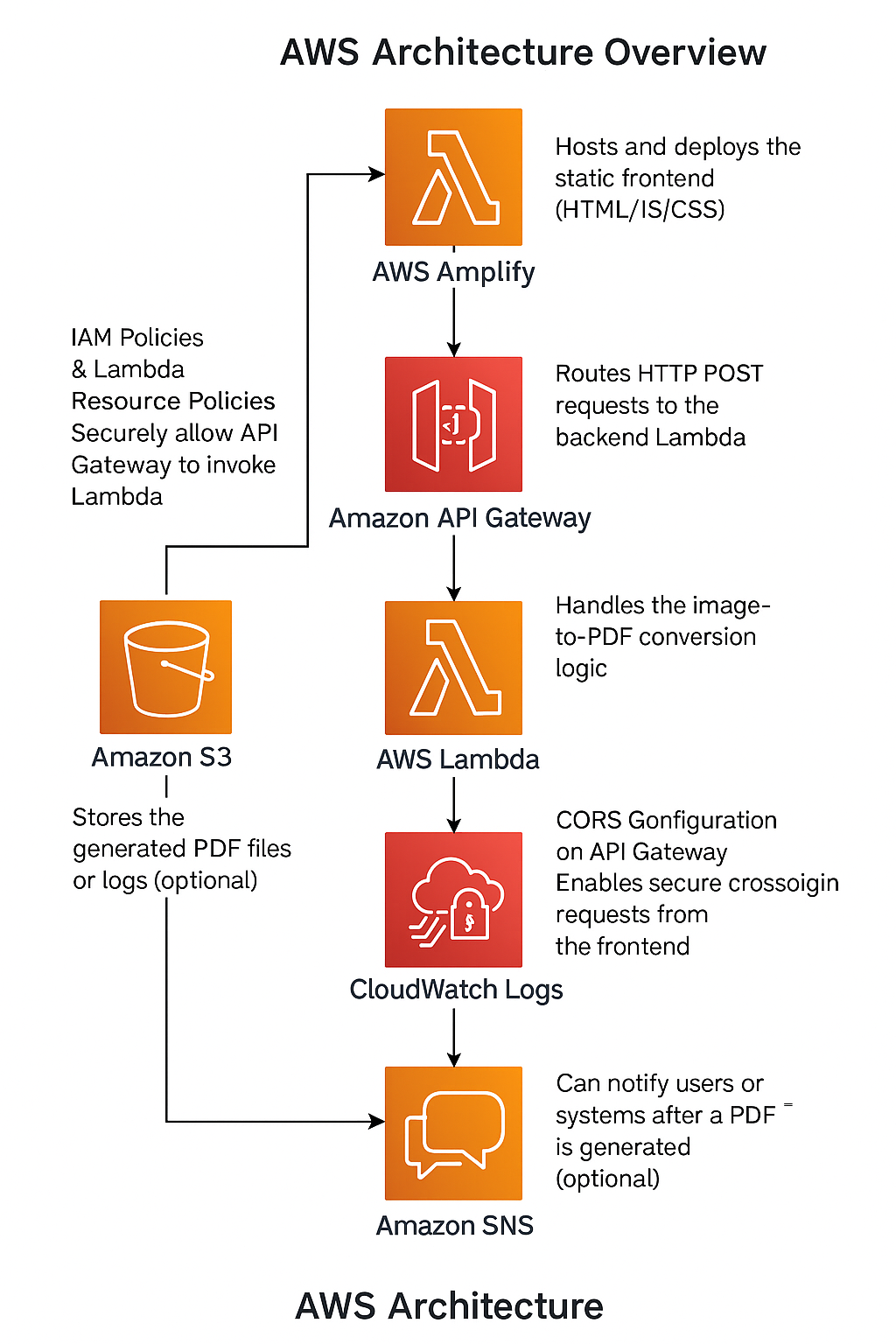
* Allow users to upload and preview multiple images via a browser.
* Convert uploaded images into a single PDF using cloud-based processing.
* Use serverless infrastructure to ensure scalability, low cost, and ease of maintenance.
* Securely store the resulting PDF files and monitor application activity.

1. **AWS Services Used**

| **AWS Service** | **Purpose** |
| --- | --- |
| **AWS Lambda** | Image-to-PDF processing logic executed serverless |
| **Amazon API Gateway** | Exposes REST endpoint to trigger Lambda function |
| **Amazon S3** | Stores output PDF files and logs |
| **AWS Amplify** | Hosts the frontend (HTML/CSS/JS) |
| **IAM** | Controls access between API Gateway, Lambda, and S3 |
| **CloudWatch** | Logs and monitors Lambda execution |

**5. Architecture Diagram**

The application is designed using the following architecture:



**6. System Workflow**

1. **Frontend (User Interaction):**  
   Hosted on Amplify, allows users to select and preview images.

A screenshot of a computer

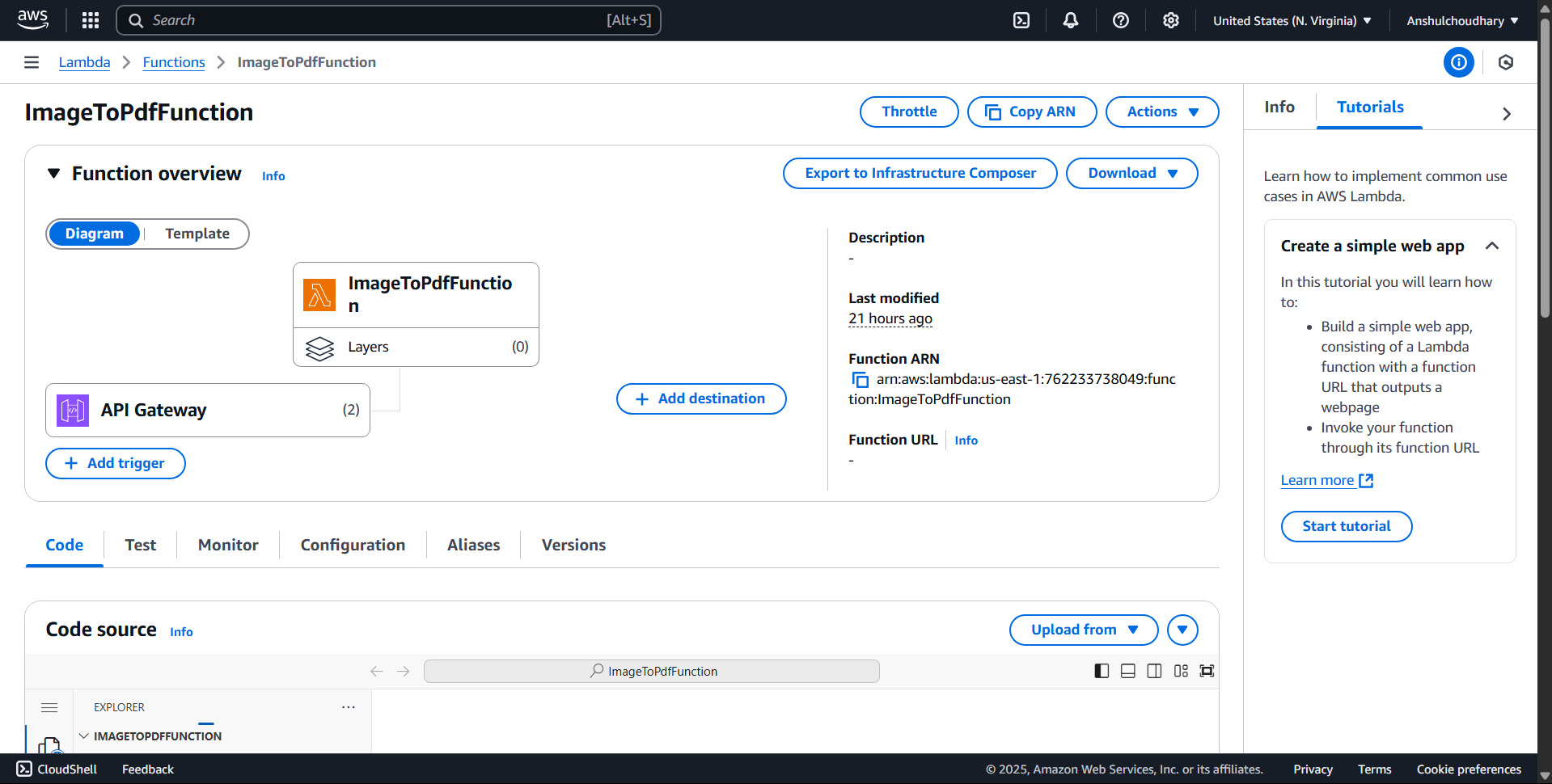
AI-generated content may be incorrect.

1. **Backend (Lambda Trigger):**  
   On submission, images are sent to an API Gateway endpoint.

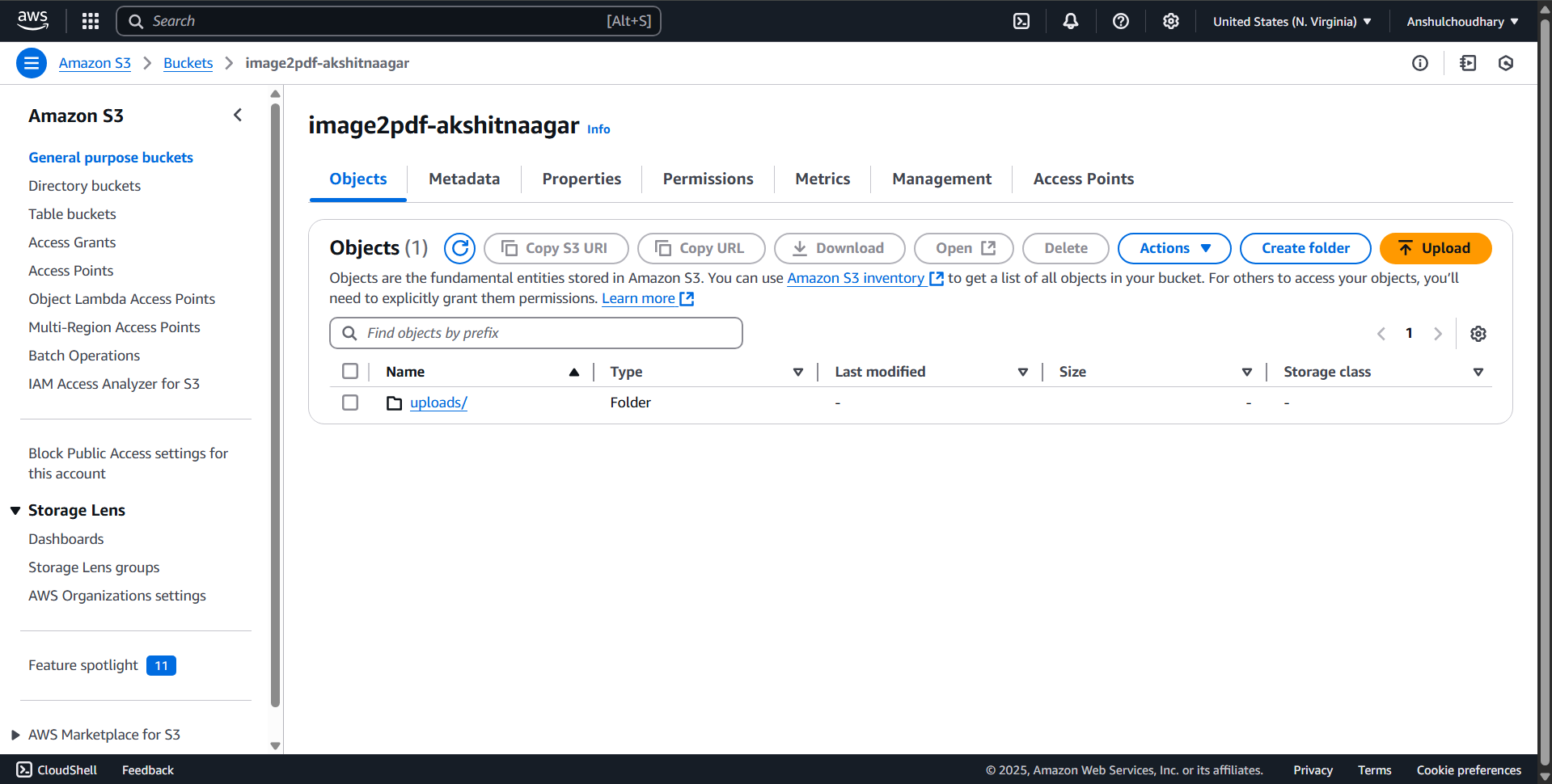
A screenshot of a computer

AI-generated content may be incorrect.

1. **Image Processing:**  
   The API Gateway invokes a Lambda function that converts the images into a single PDF.



1. **PDF Storage:**  
   The generated PDF is stored in Amazon S3.



1. **Status Response:**  
   A success or error message is sent back to the frontend.

A screenshot of a computer

AI-generated content may be incorrect.

**7. Project Directory Structure**

plaintext

Copyedit

📦 image-to-pdf-converter

┣ 📁 frontend/

┃ ┣ 📄 index.html

┃ ┣ 📄 style.css

┃ ┗ 📄 backend.js

┣ 📁 lambda/

┃ ┗ 📄 image\_to\_pdf.py

┣ 📄 README.md

**8. Deployment Steps**

* **Lambda Function:**  
  Deploy image\_to\_pdf.py through the AWS Lambda Console with proper IAM execution roles and S3 access.
* **API Gateway:**  
  Create an HTTP POST endpoint integrated with Lambda and configure CORS.
* **Frontend (Amplify):**  
  Deploy the static files (index.html, style.css, backend.js) through AWS Amplify.
* **Testing:**  
  Use a browser or Postman to test the workflow.

**9. Security Considerations**

* CORS restricted to the Amplify domain.
* File type and size validation implemented in Lambda.
* IAM roles and resource policies set with the principle of least privilege.
* Optional usage plans or API keys can limit abuse.

**10. Testing Summary**

| **Test Case** | **Result** |
| --- | --- |
| Upload multiple images | ✅ Passed |
| PDF generated successfully | ✅ Passed |
| Invalid file type rejected | ✅ Passed |
| Large image timeout handled | ⚠️ Partial (can improve with longer Lambda timeout) |
| Error message on failure | ✅ Passed |

**11. Challenges Faced**

* Handling CORS configuration properly across services.
* Image encoding and ordering before PDF conversion in Lambda.
* Ensuring correct IAM roles to allow secure access between services.

**12. Conclusion**

This project demonstrates how a full-stack application can be developed and deployed using serverless technologies. The use of AWS services like Lambda, API Gateway, and Amplify reduces the need for infrastructure management and improves scalability. It is a cost-effective, user-friendly solution for real-world document processing

**13. Future Improvements**

* Add user authentication with AWS Cognito.
* Provide downloadable links or email PDF results.
* Support for more image formats and drag-and-drop UI.
* Add database support (e.g., DynamoDB) to log user actions and analytics.