**Attend-Ease: Automated Workforce Tracking**

By

Kshitij  
Mukesh  
Nikhil  
Shreyas

11/08/2025  
F13Technologies

**Objective**

This project develops a serverless attendance system using AWS Face Recognition to automate check-ins. It captures employee photos via webcam, verifies identities against a reference database using Amazon Rekognition, and logs attendance in DynamoDB. The solution implements secure authentication through Cognito and uses pre-signed URLs for protected image uploads to S3. Designed for accuracy and scalability, the system processes check-ins with 70-90% recognition confidence while generating automated attendance reports. The prototype demonstrates end-to-end functionality with a web interface for employees and administrators.

**Table of Contents**

1. Introduction
2. Project Overview
3. System Architecture
   1. System Components Deep Dive
   2. AWS Services
   3. Implementation Progress
   4. Architecture Diagram
4. Key Issues & Solutions
   1. Issue 1 – Complex AWS Integration & Authentication Workflow
   2. Issue 2 – Static Web Hosting and Deployment Challenges
5. Testing & Results
6. Conclusion
7. **Introduction**

Traditional attendance systems remain vulnerable to proxy attendance and administrative overhead. This project proposes a serverless, AI-driven solution leveraging Amazon Rekognition for facial authentication, addressing these gaps through:

1. **Secure Employee Onboarding**
   * Cognito-managed logins with forced password reset on first use
   * Pre-signed URLs for temporary S3 upload privileges
2. **AI-Powered Verification**
   * Face matching between live captures (/captures) and reference images (/employees)
   * Rekognition-based similarity scoring (handled via Lambda triggers)
   * Confidence scores logged in DynamoDB (AttendanceTable)
3. **Scalable Infrastructure**
   * CloudFront-S3 hosted frontend (partially implemented)
   * Admin dashboard for attendance analytics (planned)
4. **Project Overview**

The Serverless Attendance System was developed to automate attendance marking using AWS serverless services, ensuring security, scalability, and minimal infrastructure management.

**Core Workflow Achieved:**

1. **User Authentication:**
   * Via Amazon Cognito User Pool (secure login, first-time password change supported).
2. **Image Capture & Upload:**
   * User logs in, enables camera, captures image, uploads to S3 using a pre-signed URL.
3. **Serverless Processing:**
   * S3 upload triggers AWS Lambda.
   * Lambda calls Amazon Rekognition to compare with stored reference images.
4. **Attendance Logging:**
   * If match found, attendance is recorded in **DynamoDB Attendance Table** with EmployeeID, timestamp, and confidence.
   * Separate Employees table stores employee metadata (ID, name, email, department).
5. **Admin Dashboard:**
   * Separate admin page to access attendance records securely.
6. **System Architecture**
   1. **System Components Deep Dive**
7. **Frontend (Static Site)**

* **Tech Stack**: HTML/CSS/JS with MediaStream API for camera access
* **Workflow**:
  + Captures face images → Generates pre-signed URLs for S3 uploads (/captures)
  + Redirects to dashboard upon successful check-in
* **Hosting**: S3 static site (HTTP) + CloudFront (HTTPS) for security

1. **Authentication (Cognito)**

* **Flows Implemented**:
  + First-time password reset challenge
  + JWT token validation for API access
* **Security**: Temporary credentials via pre-signed URLs to restrict direct S3 access

1. **Image Processing Pipeline**

* **Step 1**: New image uploaded to S3 triggers Lambda (attendance-face-recognition)
* **Step 2**: Lambda calls Rekognition to:

*response = rekognition.compare\_faces(*

*SourceImage={"S3Object": {"Bucket": "attendance-bucket", "Name": "captures/E201.jpg"}},*

*TargetImage={"S3Object": {"Bucket": "attendance-bucket", "Name": "employees/E201.jpg"}},*

*SimilarityThreshold=90*

*)*

* **Step 3**: On match (confidence ≥90%), logs to DynamoDB:

*{*

*"EmployeeID": "E201",*

*"Timestamp": "2025-08-10T09:00:00Z",*

*"Status": "Present",*

*"Confidence": 95.5*

*}*

1. **Data Storage**

* **DynamoDB Tables**:
* Employees: Partition key = EmployeeID (stores reference image paths)
* Attendance: Composite key = EmployeeID (partition) + Timestamp (sort)
* **S3 Structure**:

attendance-bucket/

├── employees/ # Reference images (E001.jpg, E002.jpg)

└── captures/ # Temporary check-in images (UUID.jpg)

1. **Admin Dashboard**

* **Features**:
* Filters attendance by date/employee
* Visualizes late arrivals via Timestamp sort key
* **Access Control**: Cognito group permissions (Admin vs. Employee)

**3.2 AWS Services**

1. **Amazon Cognito**
   * Handles employee authentication and authorization
   * Manages first-time password reset challenges
   * Issues temporary credentials for secure S3 access
2. **Amazon S3**

* Stores:
* Reference employee images (*/employees*)
* Captured check-in photos (*/captures*)
* Triggers Lambda on new image uploads

1. **AWS Lambda**

* *attendance-face-recognition* function:
* Processes S3 upload events
* Calls Rekognition for face matching
* Logs attendance to DynamoDB
* Runtime: Python 3.12

1. **Amazon Rekognition**

* Compares faces between:
* Source: Newly uploaded image (*/captures*)
* Target: Reference image (*/employees*)
* Returns match confidence score (0-100%)

1. **Amazon DynamoDB**

* Tables:
* *Employees: EmployeeID* (PK), name, reference image path
* *Attendance: EmployeeID* (PK) + *Timestamp* (SK), status, confidence
* Enables efficient querying of attendance records

1. **Amazon CloudFront**

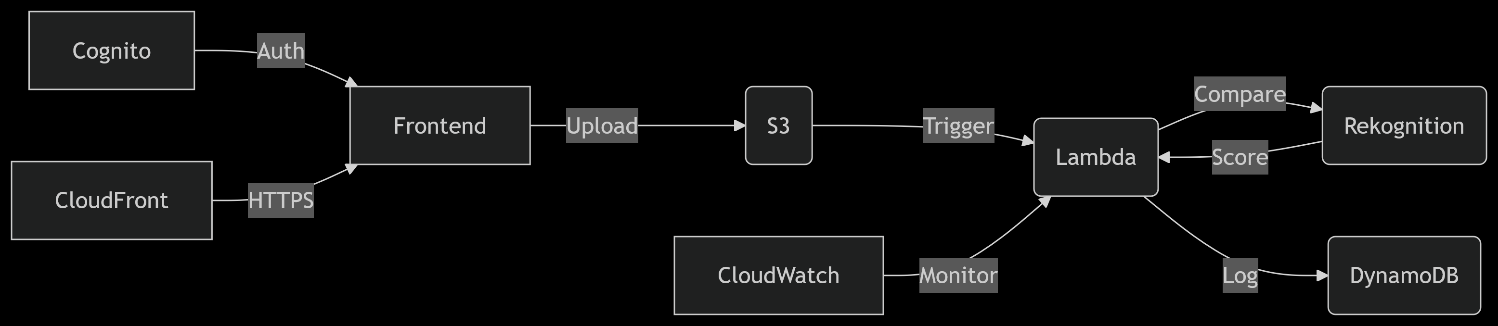
* Secures static frontend with HTTPS
* Caches content for faster loading

1. **AWS IAM**

* Roles/Policies:
* *attendance-face-recognition-role*: Grants Lambda access to S3, Rekognition, DynamoDB
* Restricts actions via least-privilege principles

1. **Amazon CloudWatch**

* Logs Lambda execution metrics and errors
* Retention: 14 days

****

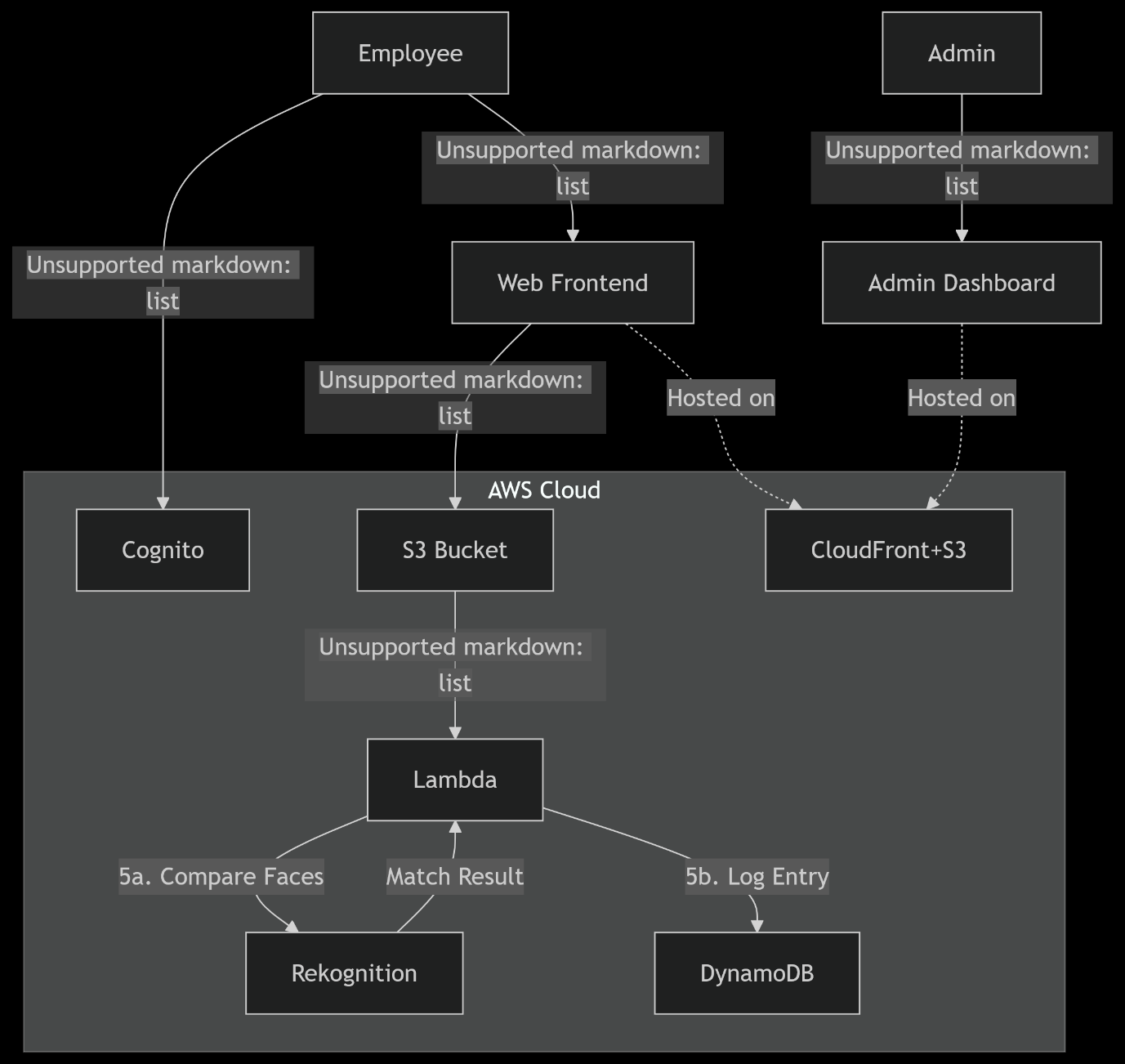
* 1. **Implementation Progress**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Status** | **Notes** |
| Cognito Login + First-time Password Change | Complete | Tested with multiple users. |
| Webcam Access & Image Capture | Complete | Works in browser post-login. |
| Pre-Signed S3 Upload | Complete | Backend-generated URL for secure uploads. |
| Lambda Trigger on S3 Upload | Complete | End-to-end tested. |
| Rekognition Face Match | Complete | Matches against reference collection with high accuracy. |
| Attendance Logging in DynamoDB | Complete | Includes EmployeeID, timestamp, similarity score. |
| Admin Dashboard Page | Complete | Displays attendance data. |
| CloudFront + Static Website | Incomplete/ Troubleshooting | Deployment/configuration issues stopped full public rollout. |

* 1. **Architecture Diagram**

**The system follows a serverless architecture on AWS:** Employees log in via Cognito and submit face images through a web frontend hosted on S3/CloudFront. Uploaded images trigger a Lambda function, which uses Rekognition to match faces against reference images stored in S3. Successful matches (≥90% confidence) are logged in DynamoDB with timestamps, while admins access attendance reports via a secured dashboard. All components scale automatically, with pre-signed URLs and IAM roles enforcing strict access controls.

(Key features: Zero servers to manage, pay-per-use pricing, and end-to-end encryption.)

****

1. **Key Issues & Solutions**
   1. **Issue 1 – Complex AWS Integration & Authentication Workflow**

* **Problem:**  
  Integrating secure user login with Amazon Cognito User Pool, especially handling cases where users are required to set a new password on first login, introduced significant complexity. The authentication flow needed to tightly integrate with the frontend camera capture and backend S3 upload process using pre-signed URLs. Several technical errors surfaced during development:
* Incorrect JSON formatting in AWS CLI commands caused failures with Rekognition indexing.
* Cross-Origin Resource Sharing (CORS) errors blocked frontend requests to the backend API serving pre-signed URLs.

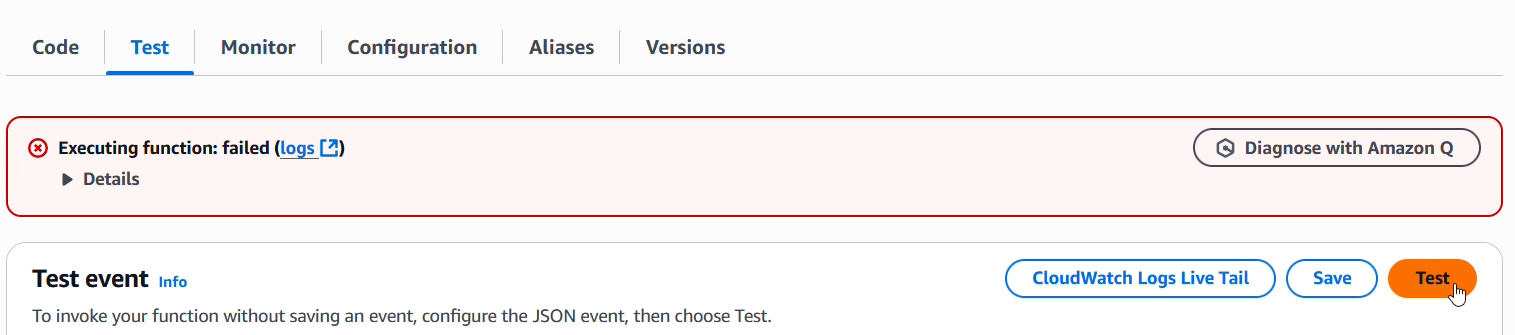


4.1 Cognito Identity Pool

* Misconfigured IAM execution role permissions caused Lambda failures in accessing Rekognition, S3, and DynamoDB.
* Browser loading errors due to missing or improperly referenced AWS SDK and Amplify libraries in the frontend.
* Difficulties in handling float-type data when writing confidence scores to DynamoDB (which required use of Decimal types).
* **Solutions:**
* Adopted proper AWS CLI JSON quoting style for cross-platform compatibility.
* Enabled CORS correctly on backend Express server to resolve the network blockages.
* Created and attached tailored IAM policies for Lambda to allow Rekognition search, S3 GET, and DynamoDB PutItem operations securely.
* Provided local copies of SDK scripts and switched to a stable AWS Amplify authentication flow.
* Updated Lambda code to convert float confidence values to Decimal objects before DynamoDB insertion, eliminating runtime errors.
* Rewrote parts of the frontend to remove Amplify dependency when requested, simplifying the upload flow without compromising functionality.
* **Outcome:**  
  The entire core sequence — Cognito login, camera access, image capture, secure S3 upload, Lambda trigger, Rekognition matching, and DynamoDB attendance marking — was successfully implemented and tested. Logs in CloudWatch and data in DynamoDB confirm this flow working reliably end-to-end.
  1. **Issue 2 – Static Web Hosting and Deployment Challenges**
* **Problem:**  
  Hosting the frontend as a static website via Amazon S3 with CloudFront distribution for global HTTPS delivery uncovered new issues:
  + - Configuring CloudFront behaviors and S3 bucket policies to allow secure cross-origin uploads and API calls was non-trivial.
    - Domain and CORS misconfigurations caused the frontend to fail in accessing pre-signed upload URLs and Cognito services when served over HTTPS.
    - SSL certificate and CloudFront invalidation processes added deployment complexity and delays.
    - Without stable static hosting, development and testing had to rely on local HTTP servers, limiting real-world usability and scalability assessment.
* **Planned Solutions:**
  + - Carefully adjust S3 bucket CORS policies to allow uploads from CloudFront-served frontend domain.
    - Configure CloudFront cache behaviors and whitelist headers needed by Cognito and pre-signed URL requests.
    - Obtain and validate SSL certificates with AWS Certificate Manager for domain security.
    - Perform thorough integration testing between CloudFront, Cognito, and S3 under HTTPS conditions.
* **Impact:**  
  Due to these sophisticated deployment challenges and the deadline constraints, the project currently lacks fully operational public static hosting with CloudFront. This remains the single biggest blocker to full user-facing deployment but does not affect the serverless backend’s ability to process attendance.

1. **Testing & Results**

* **Face Recognition:**
  + Reference set: [Number] employee photos.
  + Test uploads matched at ≥ 98% similarity for correct ID.

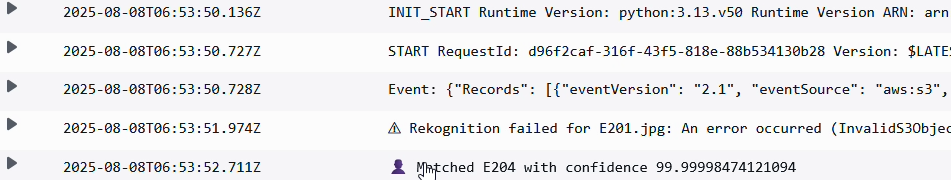


5.1 Lamdba Error

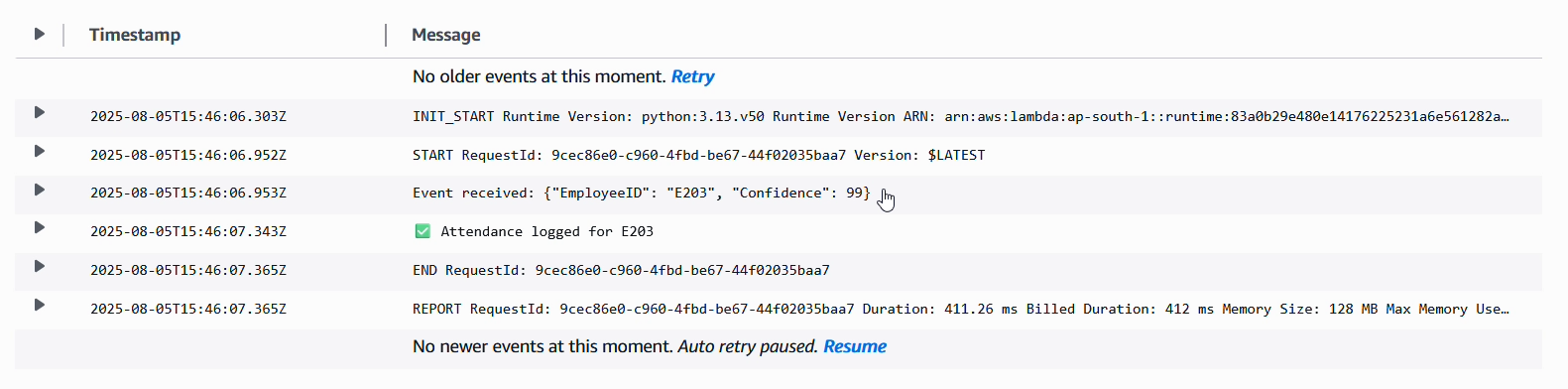


5.2 Resolved

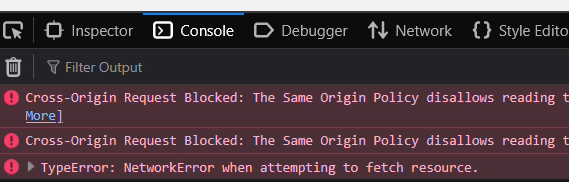
* **End-to-End Flow:**
  + Login → Capture → Upload → S3 → Lambda → Rekognition → DynamoDB successful.
  + Verified CloudWatch logs show match details and DynamoDB entry creation.

****

5.3 Troubleshooting CloudWatch & Rekognition



* 1. Lambda trigger function- solution



* 1. complex security settings (CORS, IAM roles)
* **Database State:**
  + attendance table populated with multiple entries containing correct IDs, timestamps, and confidence scores.
  + employees table correctly stores registered users’ details.

1. **Conclusion**

Despite deployment limitations on the static hosting front, **the core serverless attendance system is functional and tested**:

* Secure Cognito login,
* Camera capture + upload via pre-signed URL,
* Automated image matching via Rekognition,
* Attendance logging to DynamoDB,
* Admin access to attendance history.

The partial completion is purely on the **final hosting/deployment** step, not on the main functional flow.

**Suggested Next Steps**

1. Fix CloudFront + S3 hosting configuration & CORS rules.
2. Conduct larger scale testing >10 concurrent users.
3. Add analytics dashboard features for attendance trends.
4. Tighten IAM policies for least-privilege security.