Real-Time Threat Detection using AWS Microservices

GuardDog

Eashan Kaushik, ek3575
Rishab Singh, rs7623
Vedang Mondreti, vm2129
Vishal Prakash, vp2181

Presentation - Link
Demo- Link
GitHub – Link

ABSTRACT

As a part of the course Cloud Computing and Big Data Systems (CS-GY 9223), we have implemented a home security system that is designed to be scalable, reliable and flexible as the user base increases. These have been achieved with the help of various cloud based infrastructure provided by Amazon Web Services. Using a laptop and webcam, we were able to achieve the project's main objective, hence teaching us how to apply theoretical knowledge to a practical real-world scenario. Through this project report, we will walk you through the motivation behind implementing this solution, the architecture and concepts that have gone into the implementation, and finally the expected behavior of the solution.

1. PROBLEM

With the rise of technology, cheaper computing power has enabled the automation of various activities in our lives. One area that has seen considerable development has been that of security. While advanced security systems have historically been expensive and used by larger organizations, they are becoming a common sight in our homes due to their wider availability and cheaper maintenance. To protect a premises, you can either have personnel on the ground or have automated systems that use cameras and sensors. The issue with the former is that it includes the human element that brings in a host of other responsibilities such as ensuring their safety, scheduling shift, maintaining payroll and so on. With an automated system, we not only avoid the need for human intervention but are also able to tailor the system to our needs. Our implementation shows how one can develop their own solution with the use of third party cloud infrastructure and their laptop's webcam while at the same time reducing the costs incurred.

2. SOLUTION

Our solution involves a laptop's webcam for monitoring intrusion into a room. The webcam is always on and as soon as a person enters its frame, the system will identify whether the person is a pre-saved known face or an unknown face. If it is a known face, no further action is taken. However if it's an unknown face, an alert is sent to the homeowner via email. If the intrusion continues, alerts are sent to the homeowner at regular intervals. We have a web interface that the user can use to manage their system. The homeowner can add and delete known faces and additionally assigned labels to faces such as a person's name. They additionally have a page that contains the logs of activity over a span of time.

3. PRODUCT FEATURES

- Intruder Alerts
- Signed URL for video access
- Person and Face detection
- Facial Recognition

4. ARCHITECTURE

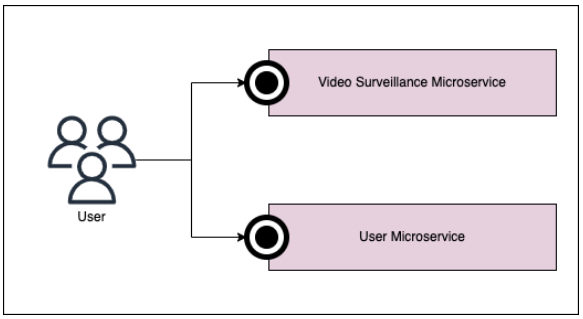

Figure 1. User Interaction

4.1 Overview

Our solution mainly has two lines of operation, one that allows the user to upload faces and associated tags while the other is the functionality of actual video capturing and recognition.

As shown in the diagram, the branch on the top represents the upload faces functionality. When the user uploads an image, a presigned URL is created that directs to a S3 bucket for storage. The pre-signed URL is a link to a S3 bucket along with an authentication key that is appended to it. Hence only someone with a pre-signed key is allowed to upload an image to the bucket. This was done to avoid public access to the images stored in the bucket which are considered confidential. Additionally, pre-signed URLs can be used to take advantage of transfer acceleration when the application scales.

Amazon S3 Transfer Acceleration is a bucket-level feature that enables fast, easy, and secure transfers of files over long distances between your client and a bucket. Transfer Acceleration is designed to optimize transfer speeds from across the world into S3 buckets by taking advantage of the globally distributed edge locations in Amazon CloudFront. As the data arrives at an edge location, the data is routed to the bucket over an optimized network path.

Once the image is uploaded to the bucket GuardDog-Known-Faces (B2), a PUT trigger invokes a lambda function GuardDog-Create-Collection-KnownFace-Recognition (LF7) that transfers the image, associated label and user information into Rekognition that is being used for image recognition. The user information is required in order to decide which collection to upload the image to; each user of the system gets their own collection and each of those collections contains the images that particular user has uploaded. Within a Rekognition collection, information such as image URL, label, bounding boxes, etc. are provided for each image (bounding boxes are automatically created by Rekognition).

The second branch of the architecture involves the use of the webcam to livestream. To enable the connection between the webcam and system entry point, we are using a C++ based plugin called Gstreamer. The livestream is sent to Kinesis Video Stream that is used as the entry point for the stream into the system. The video stream is passed on to Rekognition Video

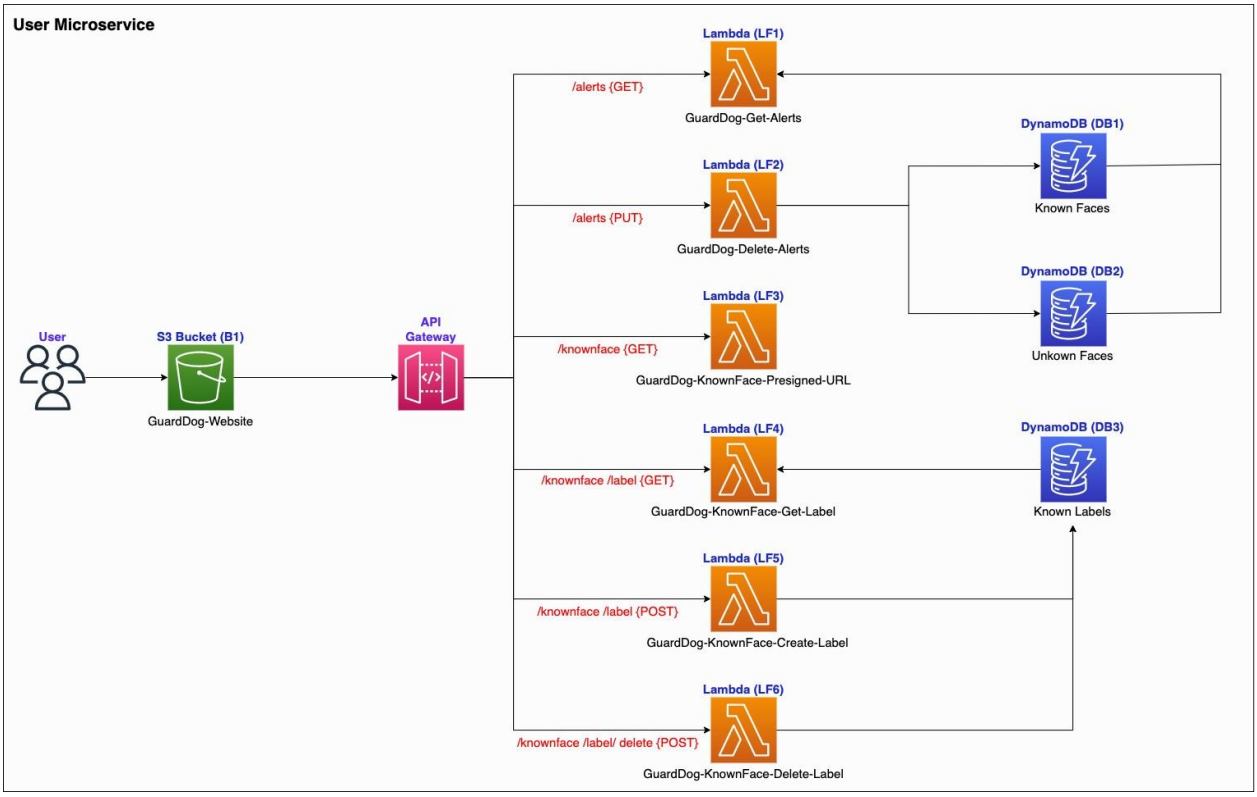

Figure 2. API Services

(RekognitionVideoLiveStream) which compares the frames of the video with the Rekognition collections that were created when the user uploaded known images. Once the analysis is done, the results are created as json files and passed on to a Kinesis Data Stream (GuardDog-Stream).

Each record in the data stream is sent to the lambda function GuardDog-Rekognition - Lambda (LF8). LF8 pushes the json record to the step function My State Machine (S1) which extracts the match condition and makes a decision based on the value. If the json record represents a match to a known face, the lambda function GuardDog-Matched-Face (LF9) is invoked which creates a log entry of the name of the face and timestamp followed by pushing it to DynamoDB Known Faces (DB1). This record is shown on the Timeline front end under the Known Faces section. If the step function receives a record that indicates an unknown face, the lambda function GuardDog-Unmatched-Face is invoked. This function performs two tasks: (1) store a log of the intrusion in the DynamoDB Unknown Faces (DB2) for publishing to the front end (under Unknown Faces in the Timeline tab) and (2) to send an email to the user about the intrusion. Since the image recognition is done for each frame in the video, a large number of log entries will be created for even small videos. To avoid overloading the database and sending out too many alerts to the user, logs of the intrusion are pushed to the

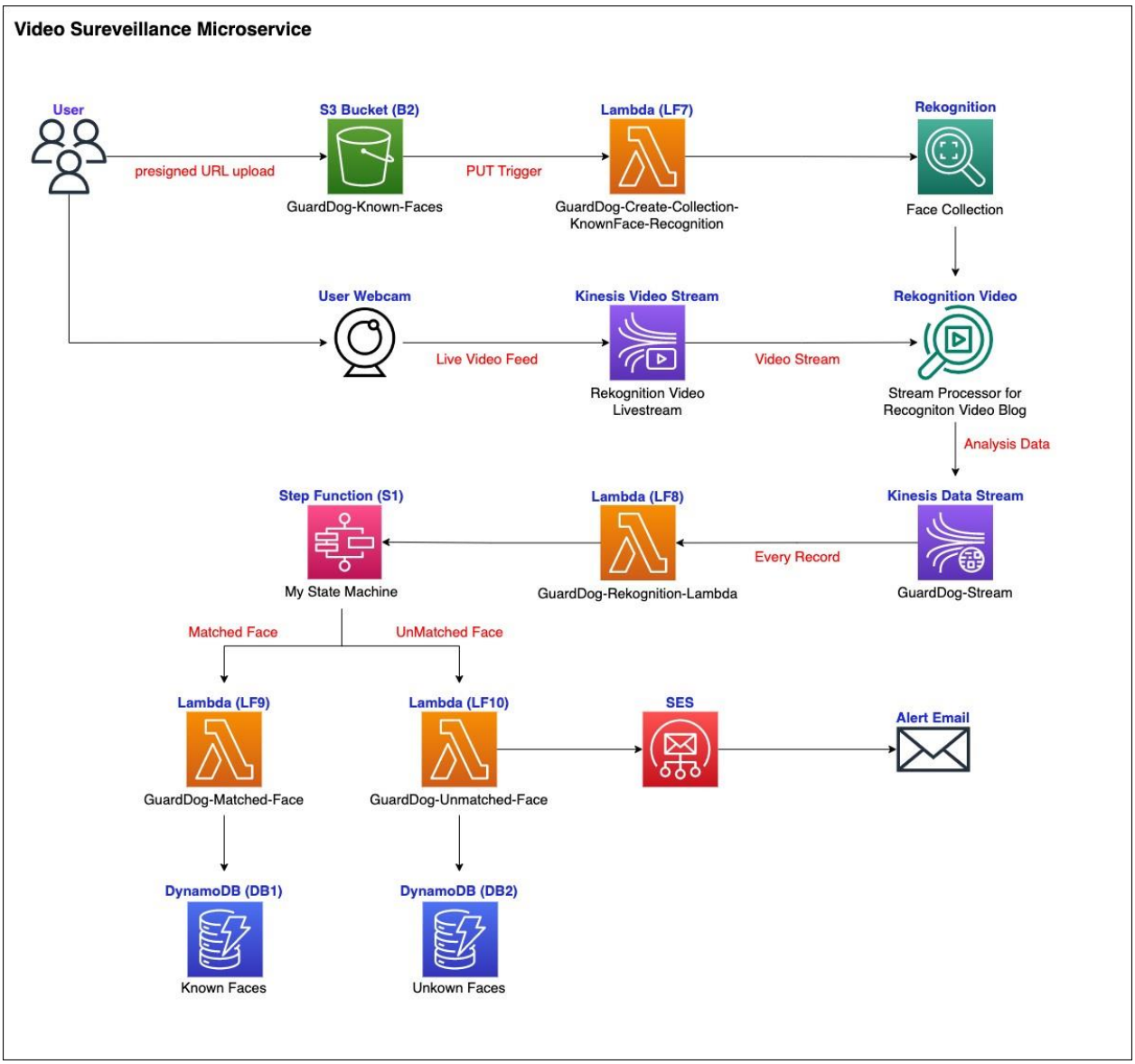

Figure 3. Architecture

database once every minute and emails are restricted to one every 10 minutes.

4.2 APIs

- **/alerts : GET**
  - Retrieves alerts from the "Known Face" database
  - Takes user_id as input
- **/alerts : PUT**
  - Deletes alerts from the "known face" and "unknown face" database
  - Takes user_id as input
- **/knownface : GET**
  - Get a presigned URL which is used by the user to upload images onto S3 bucket.
  - Takes user_id as input
- **/knownface /label : GET**
  - Retrieves the list of labels in the "Known Labels" database.
  - Takes user_id as input

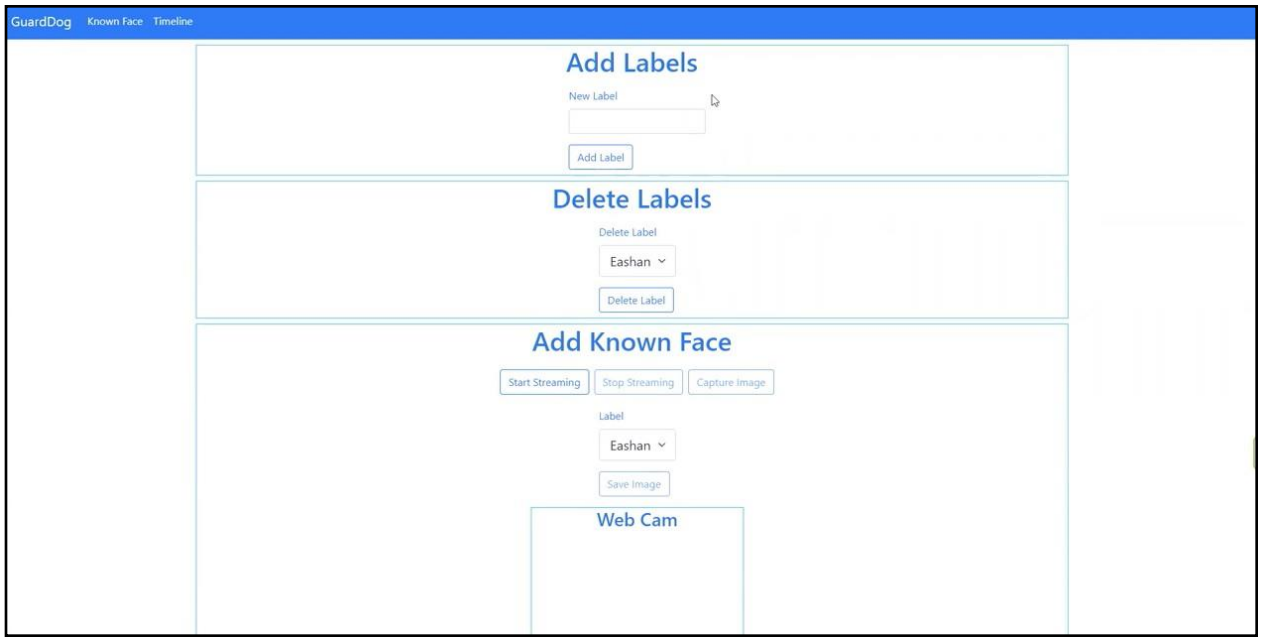

Figure 4. Front End – Known Face

- **/knownface/label : POST**
  - Create a new label for a user
  - Takes user_id and label name as input
- **/knownface/label/delete : POST**
  - Delete a label from the "Known Labels" database.
  - Takes user_id and label name as input

4.3 Databases Used

We use three DynamoDB databases to store our data. DB1 stores the records for all matched faces. Each record consists of the userid,the timestamp at which it was generated, the label of the face detected and the confidence levels. DB2 stores the records for all faces that don't match the faces in our collection. Each record consists of userid,timestamp at which the face was detected and the timestamp at which the alarm was generated. DB3 stores the list of labels for every user. A record consists of userid, timestamp at which the label was added and label name.

5. FRONT END

Our front end has two pages as shown in the screenshots.

The first page named "Known Faces" allows the user to add new and delete labels in addition to uploading an image of the known face. The Add Labels section allows the user to enter a custom label to which images can be tagged to. The Delete Labels section provides a list of existing labels in a dropdown and an option to delete the selected label. The Add Known Face section provides options to use the webcam to capture an image. There are buttons to start and stop a livestream through which the user can capture an image. The live stream is shown through the box labeled "Web Cam" and the captured image is shown in the box labeled

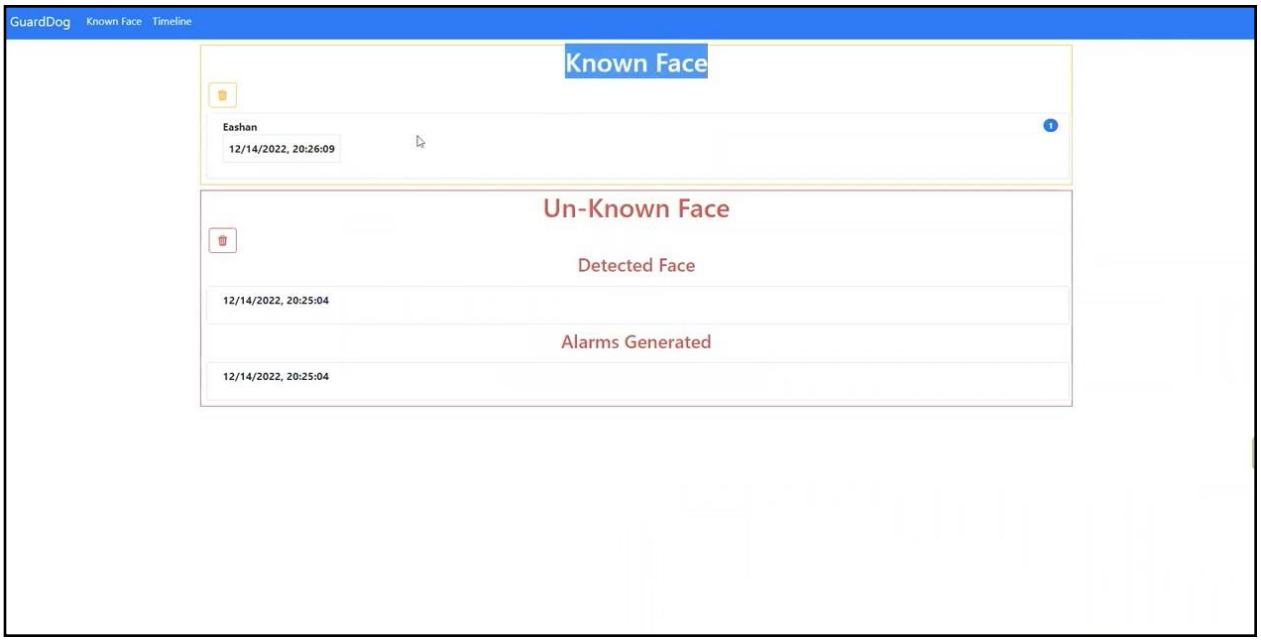

Figure 5. Front End - Timelines

"Image". A dropdown is used to tag the captured image to a pre-existing label.

The second page is called "Timeline" which contains logs of detection and alerts. The Known Face section provides the label and timestamp of a known detection while the Unknown Face provides a list of timestamps for every detection and separate timestamps for when an alert was sent to the user.