RoloBox: An Image-Aware Mobile Application using the AWS Ecosystem

Hui Li and Kenny Hunt
Department of Computer Science
University of Wisconsin-La Crosse
La Crosse, WI, 54601
{li2998, khunt}@uwlax.edu

Abstract

Several popular web applications and mobile apps support a feature that scans through a user's photo album and generates a gallery of individual faces. Photos with similar faces are grouped together, but information about the people appearing in the photos is not typically made available. RoloBox is an App that allows users to manage a list of contacts while supporting a photo-based matching feature so that users can upload images of their contacts and, more importantly, identify those contacts at a later time by taking a photo of a room or gathering with their phone. RoloBox utilizes the Rekognition API provided by Amazon Web Services (AWS) to detect and store facial features. Once a person is labeled with contact information in a photo, they will later be recognized and automatically labeled when appearing in another photo. This paper presents the design and implementation of RoloBox with an emphasis on how AWS Rekognition and Amazon Simple Storage Service (S3) services provide foundational support and how the AWS services can be used in the context of an undergraduate CS curricula to give graduates significant and relevant experience.

1 Introduction

Several popular web applications and mobile apps such as Samsung Gallery [1], iOS Photos [2] and Google Photos [3] support a feature that scans through a user's photo album and generates a gallery of individual faces. Photos with similar faces are grouped together, but information about the people appearing in the photos is not typically made available. RoloBox is an App that allows users to manage a list of contacts while supporting a photo-based matching feature so that users can upload images of their contacts and, more importantly, identify those contacts at a later time by taking a photo of a room or gathering with their phone.

1.1 Functional Analysis

When a user uploads a photo to RoloBox, the application automatically identifies all faces in the photo. Subsequently, if a face appearing in the photo is sufficiently similar to a previously labeled face, it will be automatically be labeled with that individuals contact information. If a face is not recognized, the user will be given the opportunity to label the portrait by manually entering contact information. An understanding of this basic scenario yields the following high-level requirements given as user stories.

1. As a user, I want to be able to log in, log out, and manage my contact list.
2. As a user, I want to be able to store images of groups and invidiuals in such a way that they can be easily accessible while remaining secure.
3. As a user, I want to be able to associate portrait images with contact information.
4. As a user, I want to be able to take a picture of a group or individual and have all individuals automatically recognized or labeled as not being in my contact list.

2 RoloBox

RoloBox chose to utilize a suite of Amazon Web Services (AWS) as the foundational platform on which to deliver the required system functionality. The AWS Rekognition API is used to detect, store and recognize facial features. Once a person is labeled with contact information in a photo, the Rekognition service is able to subsequently recognize and automatically label their portrait when appearing in another photo. The AWS Simple Storage Service (S3) is used as a secure and accessible data store for images and biometric facial signatures. S3 provides low-level control over access to user data and provides a widely distributed number of scalable edge nodes. The applications web API is deployed on an Elastic Cloud Compute (EC2) engine running a Node/Express server with MongoDB database.

2.1 Architectural Overview

Figure 1 shows the overall architecture of RoloBox. The client side includes both a web application meant for use on a laptop or desktop along with a mobile application. Both of these clients communicate with the RoloBox server through a a web API that has no direct interaction with the AWS platform. The application uses a Mongo/Express/Angular/Node (MEAN) stack architecture. The server is implemented as a Node/Express web API that manages all data via Mongo and provides key services through the AWS services.

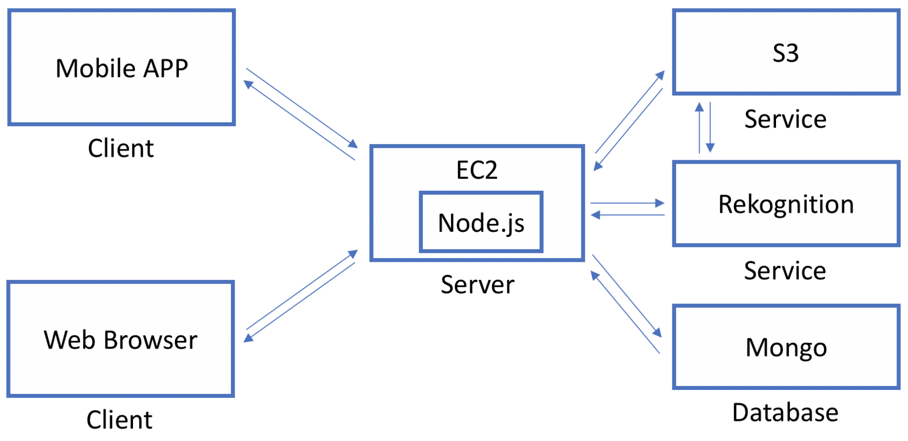


Figure 1: Application Architecture of RoloBox

3 Amazon Web Services

The World Wide Web Consortium (W3C) controls the standards that govern the internet. The W3C defines a web service as *"A software system designed to support interoperable machine-to-machine interaction over a network"* [4]. Web services can be understood as always available computational engines providing specialized functionality to authorized clients. Since interaction with these services is governed by standardized web-based protocols, the services themselves are programming language agnostic. These services are typically made available by providing bindings to these web protocols in a variety of popular languages. From a developers perspective, web services are a cost-effective way to mashup sophisticated applications from simpler, narrowly focused web services.

Web services, in general, provide the foundational infrastructure of most web applications. Amazon Web Services is the largest [5][6] and most commercially successful service [7][8] provider and one that powers a significant portion of the internet. As such, graduates of a computer science or related program of academic study are significantly advantaged with respect to the job market if they have knowledge and experience with the AWS platform. This paper overviews the use of several AWS services to create a web application of significant complexity. Although the user interface portions of RoloBox are likely beyond the

```
 1  var params = {
 2    Body: <Binary String>,
 3    Bucket: "capstone.hui",
 4    Key: "images/family.jpg"
 5  };
 6  s3.putObject(params, function(err, data) {
 7    if (err) console.log(err, err.stack); // an error occurred
 8    else     console.log(data);           // successful response
 9  });
10  /* data = {
11    ETag: "\"6805f2cfc46c0f04559748bb039d69ae\"",
12    VersionId: "tpf3zF08nBplQK1XLOefGskR7mGDwcDk"
13  } */
```

Figure 2: Usage of the AWS $putObject$ function of the S3 service

scope of most university level elective courses, the use of AWS services is relatively accessible for most Junior and Senior level CS majors. This section describes the web services used by RoloBox and gives code fragments that illustrate how these services can be used.

3.1 S3

When a user uploads a portrait of a contact, the image is saved in an AWS S3 repository. Amazon S3 uses *buckets* in which to store data *objects* where an object consists of a byte array (usually a file) along with metadata associated with that array. Buckets are the containers for objects and roughly correspond to the notion of *folders* in a traditional file system. RoloBox uses two buckets: one named *images* in which a user's photos are stored and another named *faces* in which thumbnails of contact portraits are stored.

Although the uploaded image may have a large footprint, RoloBox automatically generates a series of thumbnails for each uploaded image in order to improve the UX for all clients. Thumbnails are used to populate list-based views and, since the contact lists are often lengthy, the smaller footprint of the thumbnails reduces bandwidth requirements and improves response time.

Figure 2 shows a code fragment that makes use of the S3 JavaScript binding to store an image in S3. Lines 1 through 5 denote the parameters that are passed into the underlying web service. The $Body$ parameter denotes the binary data of the object; the Key parameter represents the path that is assigned to the object; while the $Bucket$ parameter denotes the name of the bucket containing the data object. The callback function is invoked upon completion of the $putObject$ operation and gives the client code access to information about the newly created object. This information includes the ETag value, a hash of the submitted object and a version id.

3.2 Rekognition

Amazon Rekognition is able to perform numerous computer vision related services. RoloBox leverages Rekognition in order to identify faces from images in addition to extracting coarse-level descriptive text from these images. The Rekognition service is able, for example, to determine the approximate age of an individual, whether the are wearing glasses, and whether they sport a mustache among a variety of other descriptive features.

3.2.1 Face Detection

Amazon Rekognition accepts images in both $.jpg$ and $.png$ format. The Rekognition service is exposed to developers via a JavaScript SDK. To invokes the service, we construct a data-buffer from data that originates from either a locally captured image or from an image stored on S3.

Figure 3 shows usage of the $detectFaces$ method from Amazon Rekgnition JavaScript SDK. Lines 1 to 13 define the structure of parameters to be passed into the method. The $Image$ on line 2 is passed either as base64-encoded image bytes or as a reference to an image in an Amazon S3 bucket. $Attributes$ on line 10 is an array of returned facial attributes. This can be the default list of attributes or all attributes. If no value or ['DEFAULT'] is specified for $Attributes$, the API returns a subset of all facial attributes avaiable through the API otherwise, if the value ['ALL'] is given, then all facial attributes are returned at the cost of longer time to completion. As is normally the case, the client receives the result via the callback function passed as the second argument to $detectFaces$.

```
1   var params = {
2     Image: {
3       Bytes: Buffer.from('...') || 'Base64-Encoded',
4       S3Object: {
5         Bucket: 'BUCKET_NAME',
6         Name: 'OBJECT_NAME',
7         Version: 'OBJECT_VERSION' // optional
8       }
9     },
10    Attributes: [
11      'DEFAULT' | 'ALL',
12    ],
13  };
14  rekognition.detectFaces(params, function(err, data) {
15    if (err) console.log(err.stack);  // an error occurred
16    else console.log(data);         // successful response
17  });
```

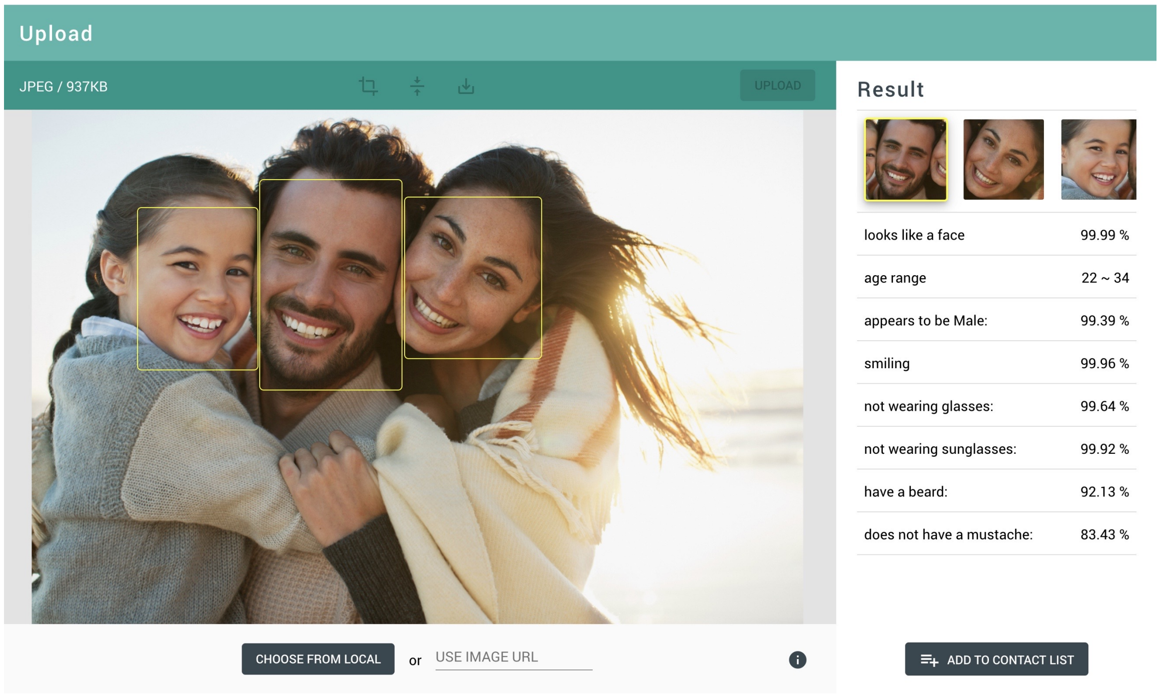
Figure 3: Usage of the AWS $detectFaces$ function of the Rekognition service

Figure 4 shows a sample face analysis result with $Attributes$ specified to ['DEFAULT'] where the default attributes include the $BoundingBox$, $Confidence$, $Pose$, $Quality$, and

Landmark features. This example provides the bounding box of the face in theh original image using normalized coordinates. The *Landmarks* attribute is used to identify the location of facial features such as the location of the left and right eyes, the nose, and terminal endpoints of a persons mouth. The *Pose* gives the rotation of the head within the image. The *Quality* setting quantifies how suitable the image is for computational analysis such that larger values of both sharpness and brightness are preferred. Note that the *Quality* setting does not necessarily correlated with an aesthetic or artistically pleasing scale. The *Confidence* level is a measure as to whether the bounding box actually contains a face. Figure 5 is a screenshot of how the results returned from Amazon are used to identify faces within a particular uploaded image.

```
{
  "FaceDetails": [
    {
      "BoundingBox": {
        "Width": 0.20394515991210938,
        "Height": 0.4204871356487274,
        "Left": 0.1556132435798645,
        "Top": 0.11629478633403778
      },
      "Landmarks": [
        {
          "Type": "eyeLeft",
          "X": 0.23311273753643036,
          "Y": 0.28700149059295654
        },{
          "Type": "nose",
          "X": 0.2994106709957123,
          "Y": 0.3559057116508484
        },
        /* ...eyeRight, mouthLeft... */
      ],
      "Pose": {
        "Roll": -3.4967963695526123,
        "Yaw": 4.581802845001221,
        "Pitch": 21.05695152282715
      },
      "Quality": {
        "Brightness": 69.75423431396484,
        "Sharpness": 95.51618957519531
      },
      "Confidence": 99.99998474121094
    }
  ]
}
```

Figure 4: Data returned from *detectFaces*



looks like a face	99.99 %
age range	22 ~ 34
appears to be Male:	99.39 %
smiling	99.96 %
not wearing glasses:	99.64 %
not wearing sunglasses:	99.92 %
have a beard:	92.13 %
does not have a mustache:	83.43 %

Figure 5: Screenshot of Detect Face Page on Web APP

3.2.2 Face Indexing and Searching

Note that $detectFaces$ is a stateless API operation, which means it does not persist any data. Faces need to be stored so that they can be used for later searching and matching. Each user is assigned a collection that holds the images for that user. After a collection is available for a user, image-related data is then stored into that collection.

Figure 6 shows how to create a collection for a user. The user's RoloBox id serves as the $CollectionId$ to identify the owner of the collection.

```
var params = {
  CollectionId: <String>,
};
rekognition.createCollection(params, function(err, data) {
  if (err) console.log(err, err.stack);
  else    console.log(data);
});
```

Figure 6: Usage of the AWS $createCollection$ function of the Rekognition service

Figure 7 illustrates the use of the $indexFaces$ method; a method that is very similar to $detectFaces$ but that stores recognized faces for later matching. In this method, $QualityFilter$ specifies a quality bar that determines how much filtering is done to identify faces. Note that Amazon Rekognition doesn't save the actual pixel data of faces that are detected. Instead, the underlying detection algorithm first detects the faces in the input image. Then, for each detected face, the algorithm extracts facial features into a feature vector which is stored in the backend database. The feature vector is a low-dimensionality representation of the facial elements found in the image but the image data itself cannot be reconstructed from this feature vector. Amazon Rekognition uses these feature vectors when it performs face matching operations.

```
var params = {
  CollectionId: 'STRING_VALUE', /* required */
    Image: { /* required */
    Bytes: Buffer.from('...') || 'Base64-encoded',
    S3Object: {
      Bucket: <String>,
      Name: <String>,
      Version: <String>
      }
    },
  DetectionAttributes: [
    'DEFAULT' | 'ALL'
  ],
  ExternalImageId: <String>,
  MaxFaces: <Number>,
  QualityFilter: 'NONE' | 'AUTO' | 'LOW' | 'MEDIUM' | 'HIGH'
};
rekognition.indexFaces(params, function(err, data) {
  if (err) console.log(err, err.stack);
  else   console.log(data);
});
```

Figure 7: Usage of the AWS $indexFaces$ function of the Rekognition service

A face ID is returned after adding a face to a collection with $indexFaces$. This facial id can be used to search for matching faces in the collection that the face belongs to with the AWS $searchFaces$ method. Figure 8 shows the usage of $searchFaces$ and the returned result. This code fragment looks in a collection of previously recognized faces (denoted by the $CollectionId$ parameter) for a face that is nearly identical to an unknown face (denoted by the $FaceId$ parameter) using a threshold of 90 (denoted by the $FaceMatchingThreshold$) in a normalized similarity metric space. This method will return, at most, the top ten most similar faces.

Figure 9 shows a screenshot of a face match when a user uploads a photo containing a facial image of someone they have previously rekognized and labeled.

```
var params = {
  CollectionId: "cff736e7-1cca-41d1-9cfd-8119aeb36eec",
    FaceId: "70008e50-75e4-55d0-8e80-363fb73b3a14",
    FaceMatchThreshold: 90,
    MaxFaces: 10
};
rekognition.searchFaces(params, function(err, data) {
  if (err) console.log(err, err.stack);
  else   console.log(data);
});
/*
data = {
  FaceMatches: [{
    Face: {
      BoundingBox: { ... },
      Confidence: 99.99949645996094,
      FaceId: "8be04dba-4e58-520d-850e-9eae4af70eb2",
      ImageId: "465f4e93-763e-51d0-b030-b9667a2d94b1"
    },
    Similarity: 99.97222137451172
  }, ...]
}
*/
```

Figure 8: Usage of the AWS $searchFaces$ function of the Rekognition service

4 Conclusion

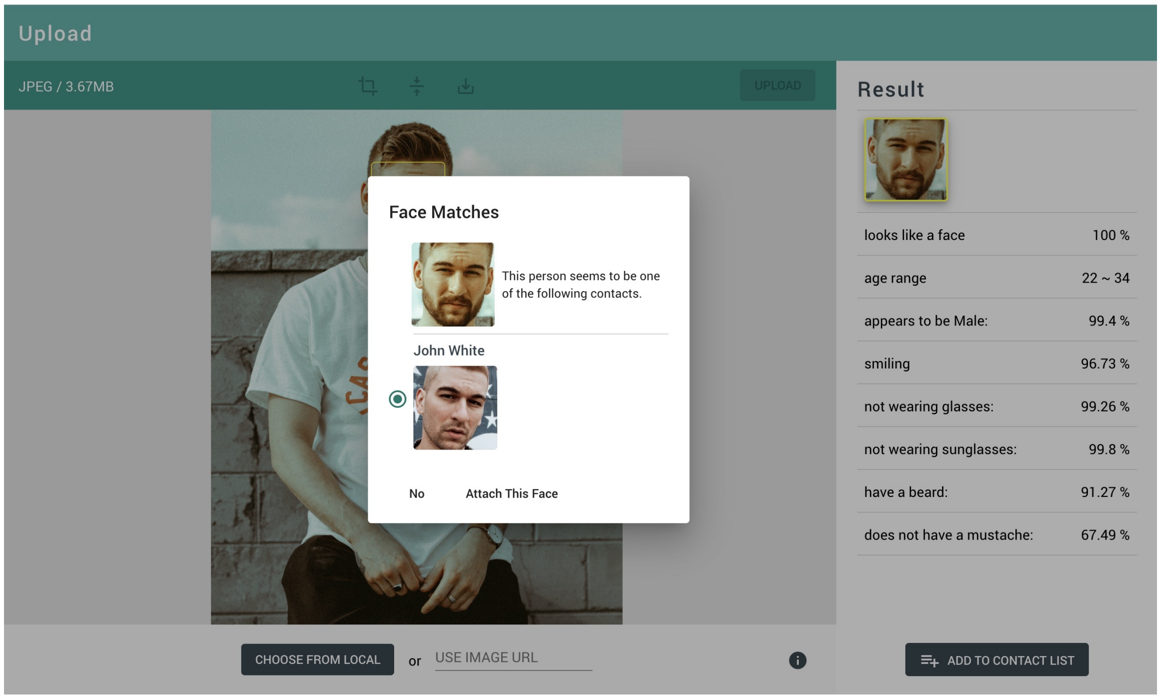
4.1 Potential Pitfalls

If the input image is in $.jpeg$ format, it might contains exchangeable image file format (Exif) metadata that includes the image's orientation. Amazon Rekognition uses this orientation information to perform image correction - the bounding box coordinates are translated to represent object locations after the orientation information in the Exif metadata is used to correct the image orientation. Images in $.png$ format don't contain Exif metadata and the value of $OrientationCorrection$ is null.

Howerver, Amazon Rekognition does not correct the image for you. $Sharp$ is a high performance Node.js image processing module and it is used in this project to perform image rotation. The code segment in Figure 10 shows how to rotate an image 90 degrees clockwise. Before sending face bounding boxes to clients, our RoloBox server determines whether to rotate the image by checking the value of $OrientationCorrection$.

4.2 Other AWS Services

AWS is known for its secure and scalable compute capacity in the cloud [9][10][12]. The Elastic Compute Cloud (EC2) service provides virtual machines that developers can fire up



Face Matches

This person seems to be one of the following contacts.

John White

No   Attach This Face

Result

looks like a face	100 %
age range	22 ~ 34
appears to be Male:	99.4 %
smiling	96.73 %
not wearing glasses:	99.26 %
not wearing sunglasses:	99.8 %
have a beard:	91.27 %
does not have a mustache:	67.49 %

Figure 9: Screenshot of Face Match

```
sharp('input.jpg')
  .rotate(90)
  .toBuffer(function (err, outputBuffer, info) {
    // outputBuffer contains rotated image data
  });
```

Figure 10: Usage of the *Sharp* library function *sharp* to rotate an image

for any purpose. We have successfully used an open source clone of the EC2 service to deploy this project as well as making extensive use of this EC2 clone in a variety of upper-level courses in our university curriculum at the University of Wisconsin La Crosse. An EC2 VM can be used not only to host web applications, but also as a way to teach system development, open source toolkits, or networking.

Rolobox currently matches faces on the mobile application by taking a photo and then searching through that static image to identify individuals. We would like to extend this technique to real-time video analysis so that the Rekognition engine results can be used to overlay streaming video. AWS provides a solution for this desired outcome via the use of the AWS Kinesis and AWS Rekognition on Video services. These services would allow a mobile phone to start the camera and send video streams to Amazon Kinesis which then creates a Kinesis video stream. The output of this Kinesis stream is fed into the Rekognition service to create a stream processor that analyzes the video stream and returns the analysis as a Kinesis data stream. Finally, the mobile phone receives this analyzed data stream and displays the recognized faces. While we have not used these services, they are likely accessible to students in a senior level elective or students involved in independent study coursework.

4.3 Suitability of AWS services in CS curricula

It is our opinion that many AWS services can be used in an undergraduate CS curricula to achieve results that are both engaging to students and extremely relevant for those same students upon entering the job marketplace. Students will likely need to be of at least Junior status in order to make effective use of the AWS SDK's. Also, while this paper presents the AWS framework through the lens of JavaScript bindings, the same services are available using Java, Python, Ruby, Node, PHP, and .NET [11].

References

[1] "Samsung Gallery". Retrieved March 27, 2020 from https://play.google.com/store/apps/details?id=com.sec.android.gallery3d&hl=en_US.

[2] "Photos for iOS and iPadOS - Apple". Retrieved from March 27, 2020 from https://www.apple.com/ios/photos/.

[3] "Google Photos". Retrieved March 27, 2020 from https://www.google.com/photos/about/.

[4] "Web Services Glossary & Web service". W3C. 11 February 2004. Retrieved March 18, 2020 from https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice.

[5] "Here is Why AWS Still Dominates the Cloud Market". Retrieved March 27, 2020 from https://www.comparethecloud.net/articles/here-is-why-aws-still-dominates-the-cloud-market/.

[6] "AWS Is Now the Largest System Business in the World". Retrieved March 27, 2020 from https://www.nextplatform.com/2019/04/30/aws-is-now-the-largest-systems-business-in-the-world/.

[7] "2019 State of the Cloud Report from Flexera". Retrieved March 18, 2020 from https://resources.flexera.com/web/media/documents/rightscale-2019-state-of-the-cloud-report-from-flexera.pdf.

[8] "AWS vs Azure vs Google Cloud Market Share 2020: What the Latest Data Shows". Retrieved March 27, 2020 from https://www.parkmycloud.com/blog/aws-vs-azure-vs-google-cloud-market-share/.

[9] Garfinkel, Simson L. 2007. An Evaluation of Amazon's Grid Computing Services: EC2, S3, and SQS. Harvard Computer Science Group Technical Report TR-08-07.

[10] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree and M. Schulz, "Exploiting Redundancy and Application Scalability for Cost-Effective, Time-Constrained Execution of HPC Applications on Amazon EC2," in IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 9, pp. 2574-2588, 1 Sept. 2016.

[11] "Set Up the AWS CLI and AWS SDKs". Retrieved March 27, 2020 from `https://docs.aws.amazon.com/rekognition/latest/dg/setup-awscli-sdk.html`.

[12] "Top 10 AWS Services You Should Know About". Retrieved March 27, 2020 from `https://www.clickittech.com/aws/top-10-aws-services/`.