Práctica Final ABP

Pedro Gómez López

Máster Universitario en Ingeniería Informática

Escuela Superior de Ingeniería

Informática

Albacete, España

pedro.gomez5@alu.uclm.es

Miguel Ángel Sánchez Cifo

Máster Universitario en Ingeniería Informática

Escuela Superior de Ingeniería

Informática

Albacete, España

miguelangel.sanchez15@alu.uclm.es

Hernán Indíbil de la Cruz Calvo

Máster Universitario en Ingeniería Informática

Escuela Superior de Ingeniería

Informática

Albacete, España

hernanindibil.lacruz@alu.uclm.es

Jorge Valero Molina

Máster Universitario en Ingeniería Informática

Escuela Superior de Ingeniería

Informática

Albacete, España

jorge.valero1@alu.uclm.es

Alejandro Moya Moya

Máster Universitario en Ingeniería Informática

Escuela Superior de Ingeniería

Informática

Albacete, España

alejandro.moya4@alu.uclm.es

Alejandro Zornoza Martínez

Máster Universitario en Ingeniería Informática

Escuela Superior de Ingeniería

Informática

Albacete, España

alejandro.zornoza1@alu.uclm.es

**Resumen**

En este documento se presenta el diseño e implementación de una aplicación en la nube que es capaz de controlar el acceso a un edificio inteligente, mediante el reconocimiento y control de las caras de las personas. El objetivo principal de este trabajo es mejorar los conocimientos de aplicaciones desplegadas en entornos cloud, mediante la elaboración de un proyecto real.

**Palabras clave**

Amazon Web Servicies; Rekognition; Reconocimiento de caras; Máquinas virtuales, Cloud, Seguridad, Domótica.

# Introducción

El objetivo principal del proyecto se basa en la creación de una aplicación que nos permita reconocer y gestionar rostros de personas, con el fin de llevar a cabo un control de acceso a un determinado lugar.

Para ello, es necesario diseñar una arquitectura que gestione las imágenes captadas por una cámara en tiempo real o de manera predeterminada.

El diseño de la arquitectura se va a basar en la utilización de los servicios que nos proporciona Amazon Web Services (AWS en adelante), dado que nos permite cubrir toda la casuística pertinente. Además, dado que se nos impone la restricción de realizar el proyecto empleando máquinas virtuales (VM en adelante), se utilizará EC2 de AWS por estar basado en instancias de VMs.

# Motivación

En España se realizan cerca de 2.36 millones de horas extra a la semana. La totalidad de estas horas no son pagadas ni compensadas con horas de descanso. Estas horas extras provoca que los trabajadores trabajen más de las 40 horas obligatorias.

El gobierno, con el fin de paliar este problema, ha llevado a cabo en un decreto de medidas urgentes de protección social y lucha contra la precariedad laboral en la jornada de trabajo [4]. En este decreto se establece la obligatoriedad de que las empresas lleven a cabo un registro de la jornada laboral efectiva de los trabajadores. Con esta medida se persigue que se paguen las horas extraordinarias y se respeten los descansos. El decreto entra en vigor a partir del 12 de mayo, dejando un margen de adaptación de dos meses, si no se cumple con lo establecido en el decreto, las empresas se enfrentan a sanciones de 626 a 6250 euros.

El problema actual es que muchas empresas no saben cómo llevar a cabo este registro, las más pequeñas lo pueden hacer en papel, pero las más grandes necesitan automatizar este servicio. En este trabajo se quiere mostrar la implementación de un sistema que permita llevar a cabo el control de las entradas y salidas del personal de una empresa. Además, el sistema desarrollado tiene una buena calidad/precio, demostrando que cumplir la ley no es costoso ni difícil.

# Componentes

A continuación, mencionaremos los diferentes componentes de AWS de los cuales hemos hecho uso para implementar nuestra aplicación:

* Amazon Cognito: Se encarga de la gestión de los diferentes usuarios que tendrán acceso al sistema. Su función principal es la gestión de identidades y autenticación en la nube de forma simple y segura
* EC2: Proporciona una amplia selección de tipos de instancias optimizados para adaptarse a diferentes casos de uso. En este componente se crearán las VMs que nos permitirán gestionar la información que será procesada por el sistema
* S3 Bucket: Almacenamiento de los objetos multimedia que tratará el sistema
* Rekognition: Encargado de procesar un streaming de video o una serie de imágenes. Una de las funcionalidades principales es la detección de caras, siendo necesario para el sistema que se va a desarrollar
* Lambda: Implementación de la gran parte de la lógica del sistema, obteniendo los resultados procesados por Rekognition y lo almacena en la BBDD
* MYSQL DB o DynamoDB. Son dos alternativas para almacenar los resultados obtenidos del servicio AWS Rekognition.

A close up of a computer

Description automatically generated

Figura 1. Arquitectura de la aplicación desarrollada

En la Figura 1 se puede apreciar como los diferentes elementos descritos se interconectan, para dar forma y funcionalidad a la aplicación que posteriormente describiremos. La arquitectura ha sido elaborada haciendo uso de la herramienta Cloudcraft [3], la cual no solo permite diagramar arquitecturas desplegables en AWS, sino que también tiene funcionalidades interesantes, entre las que se encuentran cálculo de costes relacionados con nuestros diagramas o sincronización de arquitecturas ya desplegadas en la nube.

## Visión general

La plataforma AWS se encarga de proporcionar todos los módulos que se utilizarán para desarrollar la solución. Todo se debe implementar sobre una única cuenta sobre la que se cargarán los costos de los servicios utilizados.

El primer paso para integrar el equipo es crear los usuarios que sean necesarios para que exista uno por miembro. Así, es posible controlar los permisos a los que tiene acceso cada uno de ellos. De esta forma se limitan los recursos, evitando que personas no autorizadas realicen ciertas acciones. Un buen recurso es crear grupos para asignar los mismos permisos a varias personas a la vez. Con esto es más sencillo gestionar los permisos entre varios usuarios similares.

Definir presupuestos es otra buena medida para controlar los gastos en servicios. Manteniendo una buena política de notificación de gastos se puede contener el coste de prestación de servicios.

Por otro lado, las políticas son una herramienta muy potente de AWS para asegurarse de que cierto servicio tiene únicamente acceso a otro recurso definido. Es decir, su seguridad es muy alta al definir de manera muy precisa la comunicación entre recursos. Hacer un uso adecuado de las políticas es de alta prioridad para evitar, por ejemplo, corromper información en un cubo de S3.

A su vez, los roles creados para cada servicio permiten asignar varias políticas a un único servicio o recurso. Un ejemplo es que la función Lambda tiene asignado un rol que, a su vez, contiene varias políticas (administradas por AWS y creadas especialmente para la ocasión).

El servicio Virtual Private Cloud (VPC) permite definir redes internas entre los servicios de AWS. Para que exista una conexión entre los recursos, todos ellos deben encontrarse conectados entre sí. Incluso la conexión con el exterior debe ser definida explícitamente dentro del servicio. Es bastante común que existan problemas a la hora de conectar todos los recursos utilizados, pues se utilizan conceptos de redes de computadores en la forma que AWS obliga a hacerlo.

La seguridad ofrecida a los clientes está presente en todo momento. Las contraseñas deben cambiarse nada más crear un usuario y requiere un cambio cada cierto tiempo. Sobre las contraseñas se pueden aplicar unos requisitos para asegurar que son fuertes. Además, AWS posee un rastreador que merodea por diferentes páginas web susceptibles de contener claves (como GitHub). Con este sistema se puede alertar al cliente de que sus claves han sido compartidas en público. De hecho, este es uno de los puntos clave por el que se debe evitar cualquier tipo de filtración para que nadie pueda acceder a los servicios.

La plataforma ofrece una gran cantidad de servicios que permiten implementar una solución completa sin recurrir a otros elementos externos. Los demás servicios utilizados importantes para este proyecto se describen más adelante.

## Amazon Kinesis

El módulo Kinesis de Amazon tiene por objetivo recibir flujos de información, esta información puede ser tanto de video como de datos. El sistema implementado no tiene sentido sin un flujo de video donde poder detectar las caras que pasen por delaten de la cámara o cámaras.

En esta primera versión del sistema se ha optado por el uso de una Raspberry Pi 3 Model B. Raspberry Pi es una placa computadora (SBC) muy utilizado en el Internet de las Cosas por su tamaño reducido y su bajo coste. La relación calidad precio es alta, rondando los 40 € y poseyendo las siguientes características: (1) Chipset Broadcom a 1.2 GHz con procesador ARM Cortex-A 53 de 64 bits y cuatro núcleos, (2) memoria de 1 GB LPDDR2, (3) Bluetooth v4.1 y (4) Wifi 802.11 b/g/n. Además, Raspberry permite la inclusión de módulos que permitan ampliar su funcionalidad, como la “Raspberry Pi Camera V2”, que permite incluir una cámara. Todo lo descrito lo hace ideal para nuestro sistema.

El módulo de Kinesis, tal y como está planteado en el sistema desarrollado, se puede dividir en dos partes: Kinesis Video Stream y Kinesis Data Stream. En la Figura 1 podemos ver cómo se mueve el flujo de información, desde que la imagen es captada por la Raspberry, hasta que llega la información tratada por el módulo Rekognition a la Lambda, donde reside la lógica el programa.



Figura 2. Flujo de datos de Kinesis. Extraído de [2]

### Kinesis Video Stream

Este submódulo de Kinesis es el encargado de recibir la información de video del exterior. Para ello, es necesario algún sistema informático con capacidad de captar imágenes/videos del exterior y ser compatible para enviar todo lo captado a AWS. Una solución barata es la comentada anteriormente, usando una Raspberry Pi con su módulo de cámara nativo y la instalación de las herramientas de AWS es más que suficiente para que se pueda enviar la información a AWS sin muchas complicaciones.

Para que Kinesis Video Stream funcione es necesario la creación de un *Stream* en la plataforma de Amazon donde se aloja el video recibido. Para su creación basta con ejecutar el comando siguiente:

aws kinesisvideo create-stream --stream-name **PiStream** --data-retention-in-hours **1**

Se especifica que el *Streamer* de video creado se llame “*PiStream*” y la imagen captada es almacenada durante una hora. La retención de datos es necesario para que la lambda, a la hora de detectar una cara, almacene el fotograma donde el usuario ha sido detectado. Hay que tener en cuenta que la retención de video supone un coste extra en el sistema.

Una vez creado el *Stream* de Kinesis Video Stream se procede a la ejecución del programa encargado de enviar las imágenes al *Stream* creado anteriormente. En nuestro caso, dicho programa es ejecutado en una Raspberry Pi. El programa utilizado es el propuesto por Amazon para Raspberry, un ejemplo sencillo que sirve de base para la creación de software más complejo, en nuestro caso, el ejemplo sencillo es más que suficiente. Una vez compilado el programa de ejemplo, se procede a su ejecución de la siguiente manera:

./kinesis\_video\_gstreamer\_sample\_app PiStream -w **640** -h **480** -f **15**

Este programa acepta como parámetros el nombre del *Streamer* de video, en nuestro caso “PiStream”. Además, hay que especificar el ancho (-w) y alto (-h) del video capturado y la velocidad de fotogramas (-f). Los parámetros especificados en el comando son los obligatorios para que la imagen capturada se envíe a AWS sin problemas.

Antes de ejecutar el comando anterior es necesario especificar el usuario y contraseña del usuario que realizará la conexión de entrada a Amazon, para ello se ha creado el usuario “kinesis-video-raspberry-pi-producer”. Además, hay que habilitar el controlador de la “Raspberry Pi Camera V2”. Para especificar el usuario/contraseña del usuario Productor de Amazon y habilitar el controlador se tiene que ejecutar los siguientes comandos:

export AWS\_ACCESS\_KEY\_ID=**YourAccessKeyId**

export AWS\_SECRET\_ACCESS\_KEY=**YourSecretAccessKey**

export AWS\_DEFAULT\_REGION=eu-west-1

sudo modprobe bcm2835-v4l2

Tras ejecutar el programa y accediendo a Kinesis Video Stream desde la consola de AWS, podemos visualizar el video capturado, tal y como se ve en la **¡Error! No se encuentra el origen de la referencia.**.



Figura 3. Video capturado por Kinesis Video Stream

### Kinesis Data Stream

Este submódulo de Kinesis es el encargado de recibir la información emitida por Rekognition, tal y como se puede visualizar en el apartado de Amazon Rekognition. Para entender la información que recibe Kinesis Data Stream, se recomienda leer en primera instancia el módulo de Amazon Rekognition.

Para empezar a recibir la información enviada por Rekognition, se debe de crear un *Streamer* de datos (no confundir con el *Streamer* de video creado anteriormente). La creación del Streamer se puede realizar con el siguiente comando:

aws kinesis create-stream --stream-name **FacialRecognition** --shard-count **1**

Se especifica que el *Stream*er de datos tenga por nombre “*FacialRecognition*”, además, se indica que la cantidad de fragmentos que puede recibir a la vez, en nuestro caso, dado que solo tenemos una cámara funcionando se especifica una capacidad de 1, incrementar este número supone un aumento del coste.

## Amazon Rekognition

El módulo Rekognition de Amazon tiene por objetivo la detección de personas en una imagen o conjunto de imágenes, también permite la detección de personas en un determinado video o un video captado en tiempo real. El uso que hacemos de Rekognition en el sistema implementado es la detección de personas en tiempo real. La ventaja de utilizar Rekognition frente a otras propuestas es el hecho de que el módulo de detección ya está entrenado y probado por Amazon, garantizándonos así un funcionamiento óptimo.

El sistema desarrollado plantea el funcionamiento de Rekognition de la siguiente forma: un flujo de video capturado en tiempo real entra en el módulo de Rekognition, posteriormente Rekognition busca en ese flujo de video rostros de personas y las compara con un conjunto de caras previamente almacenadas, después, Rekognition devuelve un flujo de datos con la similaridad de los rostros detectados con los rostros almacenados previamente. Los rostros almacenados previamente son almacenados en una colección de caras que posee el módulo de Rekognition, para crear dicha colección de caras se debe ejecutar el siguiente comando:

aws rekognition create-collection --collection-id **faces**

En este comando se especifica que se crea una colección de caras con nombre “*faces*”. A esta colección de caras, para que funcione como se debe, se deben de ir agregando caras. La agregación de caras debe de ser realizada de manera manual por el encargado del sistema implementado, la cara que se quiera añadir a la colección de caras tiene que estar previamente almacenada en un *Bucket* de S3. Para la agregación de una cara se emplea el siguiente comando:

aws rekognition index-faces --collection-id **faces** --image '{"S3Object":{"Bucket":"**users.pgytic.esiiab.com**","Name":"**hernanpelucas/0.jpg**"}}' --external-image-id **hernanpelucas-0**

En este comando se especifica que se añada una cara a la colección denominada “faces”. Se indica que la imagen se encuentra en el Bucket con nombre “users.pgytic.esiiab.com” y la ruta de la imagen que se quiere añadir está en “hernanpelucas/0.jpg”. Por último, se especifica un nombre único a la imagen, con el fin de facilitar a posteriori la identificación de la cara detectada, en este caso se pone por nombre “hernanpelucas-0”. Es importante que por cada persona que se vaya a registrar al sistema, se almacenen tres fotos, una de perfil, lateral izquierdo y lateral derecho, por ello, ese comando debe de ser ejecutado 3 veces cambiando los comandos pertinentes. Para llevar a cabo el registro de personas en el sistema, se utiliza una página web montada en el módulo de EC2 encargado de subir las imágenes de la persona a S3 y de su registro en la colección de caras de Rekognition.

Una vez creado la colección de caras de Rekognition, se tiene que proceder a la configuración de Rekognition. Hay que indicar cuál es el *Streamer* de video que tiene que procesar, cuál es el *Streamer* de datos donde enviar los resultados y cuál es la colección de caras a emplear. Todo ello se hace realiza con el comando siguiente:

aws rekognition create-stream-processor --input '{"KinesisVideoStream":{"Arn":"**arn:aws:kinesisvideo:eu-west-1:609360233488:stream/PiStream/1557070873757**"}}' --name **FacialRecognitionStreamProcessor** --role-arn **<role ARN>** --stream-processor-output '{"KinesisDataStream":{"Arn":"**arn:aws:kinesis:eu-west-1:609360233488:stream/FacialRecognition**"}}' --settings '{"FaceSearch":{"CollectionId":"**faces**", "FaceMatchThreshold": **85.5**}}'

En este comando se especifica el Arn del Kinesis Video Stream y del Kinesis Data Stream creado anteriormente. También se indica que la colección de caras se llama “*faces*”. Con “FaceMatchThreshold” se especifica que todo rostro presente en Kinesis Video Stream debe tener un grado de similaridad de un 85.5% con alguna de las caras almacenadas en la colección “faces” para que Rekognition afirme que se ha detectado una coincidencia. Este parámetro es importante para evitar los falsos positivos, además nos sirve como un primer filtro de caras, los demás filtros se encuentran presentes en la lambda elaborada.

Por último, solo falta poner en ejecución el Rekognition, para ello se ejecuta el siguiente comando:

aws rekognition start-stream-processor --name **FacialRecognitionStreamProcessor**

La información que arrojará Rekognition sobre el Kinesis Video Stream es un JSON, tal y como se puede visualizar en la Tabla 1. Lo más importante de todo el JSON se la parte de “MatchedFaces”, ya que se indica que caras han pasado el filtro de similaridad de 85.5 % y nos específica a qué cara almacenada se asemeja más.

Tabla 1. JSON generado por Rekognition. Extraído de [2]

|  |
| --- |
| {  "InputInformation": {  "KinesisVideo": {  "StreamArn": "arn:aws:kinesisvideo:eu-west-1:xxxxxxxxxxxxx:stream/my-stream",  "FragmentNumber": "91343852333289682796718532614445757584843717598",  "ServerTimestamp": 1521903783.723,  "ProducerTimestamp": 1521903783.589,  "FrameOffsetInSeconds": 2  }  },  "StreamProcessorInformation": {  "Status": "RUNNING"  },  .  .  "MatchedFaces": [  {  "Similarity": 88.863960,  "Face": {  "BoundingBox": {  "Height": 0.557692,  "Width": 0.749838,  "Left": 0.103426,  "Top": 0.206731  },  "FaceId": "ed1b560f-d6af-5158-989a-ff586c931545",  "Confidence": 99.999201,  "ImageId": "70e09693-2114-57e1-807c-50b6d61fa4dc",  "ExternalImageId": "nick.jpeg"  }  }  ]  }  ]  } |

## Amazon S3

El módulo Lambda de AWS se encargará de almacenar objetos, concretamente imágenes, que son necesarios para el funcionamiento del sistema. S3 permite la definición de cubos o *buckets* en los que se almacenan los objetos. Para el sistema ha sido necesaria la creación de dos cubos:

* users.pgytic.esiiab.com: cubo donde se almacenan las fotos de los usuarios que se suben a través de la interfaz web. Los usuarios cuyas fotos estén almacenadas aquí serán los que tengan acceso al edificio. Se almacenan dentro de una carpeta con el nombre del usuario
* photologs.pgytic.esiiab.com: cubo donde se almacenan los fragmentos del stream de vídeo en los que se ha detectado una cara que ha disparado el evento de abrir la puerta

La interfaz web será la encargada de escribir y borrar del cubo *users.pgytic.esiiab.com*, mientras que Rekognition será el único módulo que lea dicha información.

El módulo Lambda es el encargado de almacenar los fragmentos del vídeo en el cubo *photologs.pgytic.esiiab.com* y el único módulo con acceso de lectura al cubo es la interfaz web.

## Amazon Lambda

El módulo Lambda de AWS se encargará de identificar al usuario que corresponde con el rostro que ha detectado Rekognition, tratar dicha información en base a la última vez que ha sido detectado, obtener la imagen para almacenarla en S3 y almacenar en el registro de accesos la información necesaria si procede.

La Lambda se ejecuta cuando se publican datos al Kinesis Data Stream con nombre FaceRecognition. Para ello se le ha asociado un consumidor llamado lambda-consumer, que se encargará de transmitir a la lambda la información publicada por Rekognition.

Dicha información tiene el siguiente formato:

|  |  |  |
| --- | --- | --- |
| Campo | Valor | |
| kinesis | Campo | Valor |
| kinesisSchemaVersion | Versión del esquema de datos de Kinesis |
| partitionKey | Clave de la partición |
| sequenceNumber | Número de secuencia |
| data | Datos transmitidos codificados en Base64 |
| approximateArrivalTimestamp | Momento aproximado de llegada del mensaje |
| eventSource | Servicio que origina el evento | |
| eventVersion | Versión del evento | |
| eventID | Identificador del fragmento del flujo que contiene al evento | |
| eventName | Nombre del tipo de evento | |
| invokeIdentityArn | Rol de quien ha solicitado la información | |
| awsRegion | Región del consumidor | |
| eventSourceARN | ARN del servicio que origina el evento (del consumidor) | |

La información del campo data, una vez decodificada, tiene el mismo formato que la salida de Rekognition, mostrado en la Tabla 1. JSON generado por Rekognition. Extraído de [2].

La Lambda, sobre cada registro que le entregue el consumidor, decodificará el campo data obteniendo la salida de Rekognition. Los registros que no contengan información en el campo FaceSearchResponse serán descartados, mientras que el resto serán pasados a otra función que iterará sobre todas las caras detectadas en la imagen (FaceSearchResponse es una lista). Ahí se descartan los registros con caras que no estén en el sistema según Rekognition y se da un trato especial a los que sí lo están:

1. Se suma la similaridad de la cara detectada con todas las fotos de un mismo usuario
2. Se divide la similaridad por 3 (cada usuario tiene 3 posibles fotos)
3. Se devuelve como usuario detectado el que más similaridad tenga, estando por encima de un umbral (85%). Si no se supera dicho umbral, se ignora el registro
4. Se hace una consulta sobre un RDS que tiene como clave primaria los usuarios con acceso al edificio para obtener el timestamp de la última vez que su cara fue detectada
5. Si el timestamp no supera un umbral (20 segundos) se ignora el registro. Si lo supera se actualiza el timestamp en RDS. Si el usuario no tiene registro de último acceso, se crea
6. Sobre los usuarios que hayan pasado el filtro del timestamp, se obtiene del Kinesis Video Stream el fragmento de video en el que la cara fue detectada y se almacena en S3
7. Se almacena en DynamoDB el nombre del usuario, momento del acceso y el nombre del objeto que se ha almacenado en S3

En cada etapa se registra información en CloudWatch detallando el éxito (considerando éxito como seguir lo que se ha indicado arriba, se haya detectado o no cara) o el fracaso (excepción producida que hace que la ejecución salga del cauce indicado) de la operación.

## Amazon Cognito

Con la implementación de Cognito se pretende facilitar el acceso de los administradores a la plataforma web de gestión. La idea principal es que solo aquellos que tienen un usuario y contraseña asignados pueden administrar el servicio. Por lo tanto, está bloqueado el registro a cualquier persona externa a la organización que implemente la plataforma.

Aunque existe la posibilidad de implementar un sistema de inicio de sesión personalizado, el servicio prestado por AWS es una opción más inteligente. Cognito posee una seguridad de serie muy elevada, probablemente más que una gestión de credenciales creada desde cero. Además, las opciones de personalización e integración con cualquier software lo hacen una herramienta muy polivalente.

La gran personalización que admite lo hace perfecto para asegurarse de que los administradores usan contraseñas fuertes. Se ha configurado para habilitar el inicio de sesión mediante correo electrónico y nombre de usuario. La contraseña debe tener ocho caracteres al menos y debe incluir números, mayúsculas y minúsculas. Aunque, para facilitar el inicio de sesión durante el desarrollo no se ha activado la autenticación de doble factor, se prevé que sea obligatorio en un entorno de producción.

Todos los usuarios se introducen dentro de un mismo grupo pues, al ser todos administradores, no es necesario dividir los permisos entre ellos. Crear dicho grupo es muy rápido a través de la consola en línea y permite configurar los requisitos de acceso. Después, gracias a las claves API es posible implementarlo en cualquier solución, con cualquier lenguaje soportado. En este caso, al ser PHP el lenguaje escogido, se utiliza el SDK de AWS para Cognito tal y como se detalla más adelante.

## Amazon RDS

El objetivo principal de Amazon Relational Database Service (Amazon RDS) es configurar, utilizar y escalar una base de datos relacional en la nube. El servicio suministra capacidad rentable y escalable al mismo tiempo que automatiza las arduas tareas administrativas, como el aprovisionamiento de hardware, la configuración de bases de datos, la implementación de parches y la creación de copias de seguridad.

En nuestro escenario concreto se ha decidido usar una base de datos PostgreSQL. La funcionalidad principal de esta base de datos es almacenar con niveles de persistencia adecuados información de los rostros de los usuarios, así como la última vez que fueron detectados por nuestro sistema. Por, tanto, las actualizaciones sobre esta base de datos vendrán provenientes de la correspondiente ejecución de la lambda y de la detección de rostros conocidos.

## Amazon DynamoDB

Amazon DynamoDB es una base de datos de claves-valor y documentos que ofrece rendimiento en milisegundos de un solo dígito a cualquier escala. Se trata de una base de datos multirregión y multimaestro completamente administrada, con seguridad integrada, copia de seguridad y restauración, y almacenamiento de caché en memoria para aplicaciones a escala de Internet.

Dentro de la nomenclatura de las bases de datos, DynamoDB se encuentra dentro de la categoría de no relacionales, mejorando las prestaciones a la hora de almacenar y acceder a los datos. En nuestro caso específico, la funcionalidad asignada a Dynamo es la de gestionar datos relacionados con los logs de acceso al edificio, en concreto, la persona y momento concretos. Esta información es posteriormente consumida por la máquina virtual, donde se encuentra la interfaz web y a través de la cual es posible consultar estos datos.

## Interfaz web

La interfaz web desarrollada se basa en proporcionar una comunicación interactiva y efectiva con algunos de los componentes utilizados.

Trata de centralizar toda la información proveniente de los siguientes módulos:

* Amazon Cognito: Se encarga de la gestión de los usuarios que tendrán acceso al sistema. La interfaz web solicitará las credenciales del usuario para comprobar sus permisos de acceso al sistema.
* S3 Bucket: Mediante la alta y baja de usuarios, se realizan una serie de operaciones basadas en la subida o eliminación de fotos de perfil de S3 Bucket, respectivamente.
* Rekognition: También mediante la alta y baja de usuarios, se procederá al registro o eliminación de fotos de perfil en la colección de caras del módulo Rekognition.
* DynamoDB: Obtención de la información de accesos detectados por el módulo Rekognition, para mostrarla en la interfaz web.

Todas estas acciones han sido desarrolladas usando el SDK de PHP, el cual se basa en una biblioteca de código abierto que facilita la integración de aplicaciones PHP con diferentes servicios AWS.

A través de la comunicación con cada uno de los módulos, se puede vislumbrar el comportamiento de la aplicación web, sin embargo, vamos a explicar cada una de las posibles acciones a realizar desde la web.

Toda la comunicación realizada se ha basado en la recepción y procesamiento de mensajes JSON específicamente definidos en la documentación de AWS para el SDK de PHP.

El primer contacto con la aplicación se basa en un control de acceso que se encarga de solicitar las credenciales de usuario con el fin de permitir el acceso a la web a únicamente usuarios con permisos desde Amazon Cognitio.

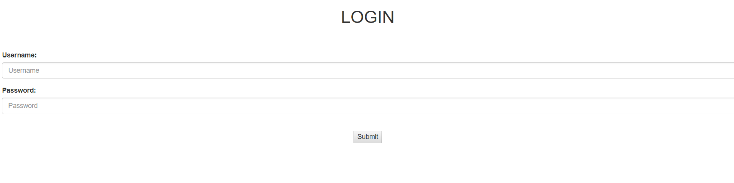


Figura 4 Control de acceso web.

Una vez accedemos a la web, podemos visualizar una tabla con los últimos accesos identificados.

En esta tabla podemos visualizar el nombre del usuario detectado y en el momento en el cual ha sido detectado.

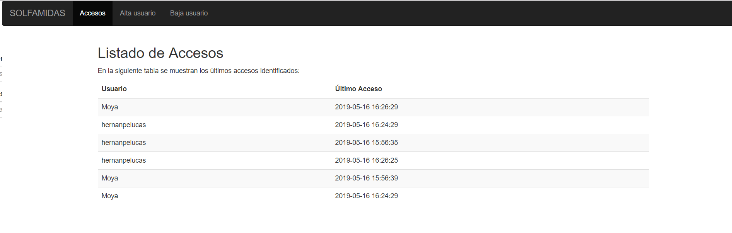


Figura 5 Tabla de accesos identificados.

Esta información se obtiene de la base de datos DynamoDB, donde se van registrando los accesos detectados por el módulo Rekognition.

Además, se incluyen dos formularios que nos permiten el alta y baja de usuarios en el sistema.

Para dar de alta un usuario, basta con incluir un nombre en el campo “Nombre” e incluir tres imágenes, las cuales corresponderán con una foto de perfil del usuario, otra del perfil derecho y otra del perfil izquierdo.

Se incorpora una limitación de imagen de 3 MB, para no sobrecargar nuestro sistema de almacenamiento con imágenes demasiado pesadas. Además, el módulo Rekognition no requiere de una resolución excesiva de imagen para realizar la detección.

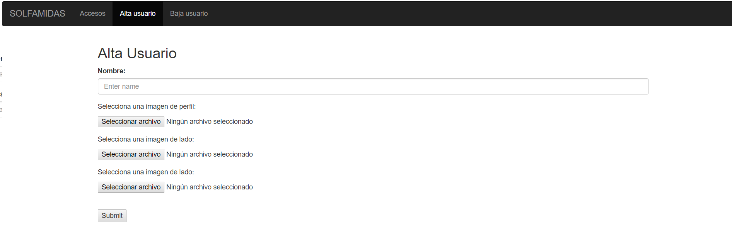


Figura 6 Formulario de alta de usuario.

Una vez hecho esto, se envía el formulario, que se encargará de la subida de dichas fotos a S3 Bucket y registrarlas en la colección de caras del Rekognition.

Por el contrario, se dispone del proceso inverso, que se encargará de dar de baja un usuario.

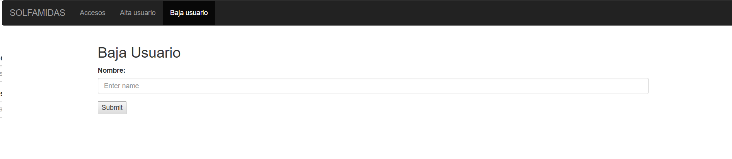


Figura 7 Formulario de baja de usuario.

En el cuál, se basa en la introducción del nombre del usuario en cuestión para proceder a eliminar sus fotos del almacenamiento en S3 Bucket, además de su eliminación en la colección de caras del Rekognition para que no vuelva a ser detectado.

# Métricas de evaluación

Las métricas que tenemos que estudiar son:

* Tiempo de despliegue
* Uso de recursos (AWS)
* Pruebas de carga (Locust)

# Costes

En relación con los costes, hemos hecho uso de la calculadora de Amazon con el objetivo de realizar una estimación aproximada de lo que costaría el despliegue de nuestra aplicación en la región de Irlanda. A continuación, describiremos dichos costes:

***Kinesis***

Datos recibidos, por GB

Primeros 500 TB/mes 0,031 USD

Siguientes 1,5 PB/mes 0,027 USD

Siguientes 3 PB/mes 0,022

Más de 5 PB/mes Contacte con nosotros

Conversión de formato de datos, por GB 0,019

***S3***

Almacenamiento estándar en S3

Primeros 50 TB/mes 0,023 USD por GB

Siguientes 450 TB/mes 0,022 USD por GB

Más de 500 TB/mes 0,021 USD por GB

Almacenamiento de S3 Estándar – Acceso poco frecuente

Todo el almacenamiento/mes 0,0125 USD por GB

Almacenamiento de S3 Única zona – Acceso poco frecuente

Todo el almacenamiento/mes 0,01 USD por GB

Almacenamiento en S3 Glacier

Todo el almacenamiento/mes 0,004 USD por GB

*Almacenamiento S3 Glacier Deep Archive*

Todo el almacenamiento/mes 0,00099 USD por GB

*S3 Intelligent-Tiering Storage, Frequent Access Tier*

Primeros 50 TB/mes 0,023 USD por GB

Siguientes 450 TB/mes 0,022 USD por GB

Más de 500 TB/mes 0,021 USD por GB

*S3 Intelligent-Tiering Storage, Infrequent Access Tier*

Todo el almacenamiento/mes 0,0125 USD por GB

*Almacenamiento S3 Intelligent-Tiering*

Monitorización y automatización, todo el almacenamiento/mes 0,0025 USD por cada 1000 objetos

***Amazon Rekognition***

*Análisis de vídeos*

0,10 USD por minuto de video analizado (prorrateado para minutos parciales).

0,12 USD por minuto de video de transmisión en directo analizado (prorrateado para minutos parciales).

*Almacenamiento de metadatos de rostros*

Precio por 1000 metadatos de rostros almacenados al mes: 0,01 USD. Los cargos de almacenamiento se aplican al mes y se prorratean en el caso de meses parciales.

***Amazon Cognito***

Capa de precios (MAU) Precio por MAU

Primeros 50 000 Gratis

Siguientes 50 000 0,00550 USD

Siguientes 900 000 0,00460 USD

Siguientes 9 000 000 0,00325 USD

Más de 10 000 000 0,00250 USD

***Amazon DynamoDB***

*Almacenamiento*

Los primeros 25 GB almacenados cada mes son gratis

• Los siguientes, 0,283 USD por GB al mes

*Solicitudes de lectura y escritura*

Tipo de cargo Precio

Unidades de solicitud de escritura 1,4135 USD por millón de unidades de solicitud de escritura

Unidades de solicitud de lectura 0,283 USD por millón de unidades de solicitud de lectura

***Amazon EC2***

CPU virtual ECU Memoria (GiB) Almacenamiento de instancias (GB) Uso de Linux/UNIX

Uso general – Generación actual

t2.micro 1 Variable 1 GiB Solo EBS 0,0126 USD por hora

***Lambda***

Memoria (MB) Segundos de la capa Precio por 100

gratuita al mes ms (USD)

128 3 200 000 0,000000208

192 2 133 333 0,000000313

256 1 600 000 0,000000417

320 1 280 000 0,000000521

384 1 066 667 0,000000625

448 914 286 0,000000729

512 800 000 0,000000834

.

2 560 160 000 0,000004168

2 624 156 098 0,000004272

2 688 152 381 0,000004376

2 752 148 837 0,000004480

2 816 145 455 0,000004584

2 880 142 222 0,000004688

2 944 139 130 0,000004793

3 008 136 170 0,000004897

***Amazon DB RDS***

*DB RDS*

Precio por hora

Instancias estándar – Generación actual

db.t2.micro 0,02 USD

# Conclusiones y trabajo futuro

Introducir aquí un párrafo de conclusiones.

# Referencias

1. Amazon Web Services, Servicios de Informática en la nube. <https://aws.amazon.com/es/sdk-for-php/>
2. Amazon Web Services. Use facial recognition to deliver high-end consumer experience with Amazon Kinesis Video Stream and Amazon Rekognition Video. <https://aws.amazon.com/es/blogs/machine-learning/improve-your-customer-service-using-amazon-kinesis-video-streams-and-amazon-rekognition-video/>
3. Cloudcraft. Visualize your cloud architecture like a pro. <https://cloudcraft.co/>
4. Real Decreto-ley 8/2019, de 8 de marzo, de medidas urgentes de protección social y de lucha contra la precariedad laboral en la jornada de trabajo.