PROYECTO ABP VER.0.2
BIG-EYE

F-TEAM

En este documento se detallarán y se mostrará la primera propuesta para el proyecto ABP sobre el procesamiento de imágenes en la nube de AWS. En la figura 1 se muestra de forma esquemática y visual como consideramos que debe quedar la arquitectura de nuestro proyecto.

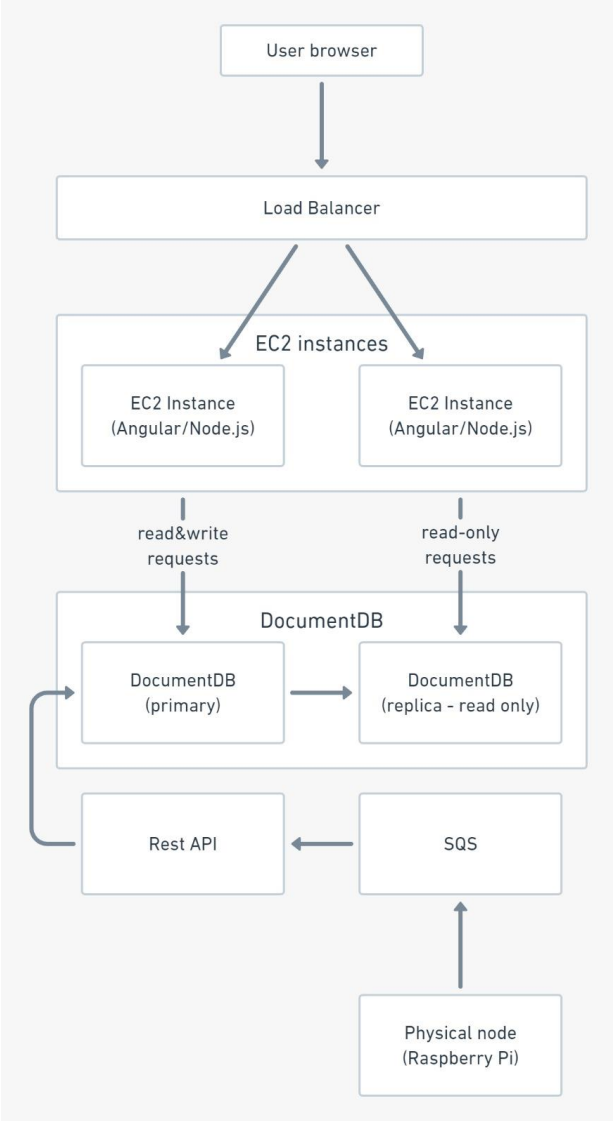

Figura 1 Estructura aplicación Big-Eye

A continuación, se van a definir y detallar cada uno de lo bloques y detalles de la arquitectura.

- **User Browser**: Es como hemos considerado representar las conexiones de los clientes/usuarios de nuestra aplicación a las instancias de Front/Back-End. Como es obvio lo que espera el usuario es poder acceder a una página web donde se represente de forma clara y sencilla la información sobre la gente que entra y sale de un determinado lugar que el mismo a solicitado. La información podría ser

mostrada a través de gráficas con estadísticas, en forma de tabla con cada entrada y salida registrada, etc. Lo ideal, sería hacerla de forma que sea perfectamente visible sin importar el dispositivo desde el que se acceder (portátil, tablet, móvil, etc.)

- **Angular/Node Web 1-2:** Estos dos componentes representarán las instancias web replicadas que usaremos para servir como servidores web a los que los usuarios se conectarán a la aplicación y podrán ver los datos pertinentes. Estas instancias estarán conectadas a las bases de datos de las cuales obtendrán la información solicitada por el usuario. Las tecnologías que se usarán son Node.js para la implantación del BackEnd y Angular para la realización del FrontEnd.
- **DB Master/Slaver:** Estructura de base de datos pensada inicialmente para tener una replicación de la información en caso de caída de servicio, sobrecarga o causa no esperada con la intención de evitar la caída del servicio y la perdida de información que puede darse por un error. También se ha considerado utilizar instancias de la base de datos que sean únicamente de lectura y que vayan obteniendo información cada X tiempo de las instancias de lectura/escritura. La primera idea es usar una DocumentDB de AWS e utilizar una MongoDB pues consideramos que para la aplicación en cuestión es mucho más interesante una base de datos NoSQL por comodidad, sencillez y compatibilidad con Node.js. Estas bases de datos recibirán los datos por parte de las imágenes procesadas de la Raspberry y enviarán la información solicitada a las instancias Web.
- **RestAPI:** Nodo que ha cambiado con la nueva versión, en lugar de realizarla sobre una instancia de EC2 vamos a utilizar las funciones Lamdas proporcionadas por AWS que nos permitirá reducir el gasto pues sólo necesitamos que esté activa cuando se invoque por parte de la raspberry pi.
- **RaspPi:** Es la representación de la RaspberryPi con la cámara que se utilizará para identificar a las personas que entran y salgan y a través la cual se procesarán las imágenes con el pensamiento de usar OpenCV (si procediera) y que cada determinado tiempo se enviase la información al Cloud. Pues por la naturaleza de la aplicación hemos pensado que no tendría mucho sentido que se realicen conexiones continuas a la base de datos.

Una vez visto los bloques de manera detallada de nuestros componentes, se van a explicar los puntos que se van a tener en cuenta tanto en el ámbito de la seguridad como escalado de la aplicación.

Como se puede ver en la figura 1, se van a implementar un load balancer para las instancias web de la aplicación. Investigando como podíamos hacer escalable también la base de datos hemos visto en la documentación de AWS que automáticamente la plataforma cloud estable un load balancer entre todas las instancias de base de datos sin importar el tipo siempre y cuando estén relacionadas. Por otra parte, para la comunicación de la RaspberryPi con las RestAPI hemos pensado que en el caso que se quiera aumentar el número de Raspberry’s y sobre todo por el tipo de comunicación que se va a realizar es mucho más interesante en cuanto a coste implementar una SQS (Simple Queue Service) en lugar de un load balancer. Esto último debemos de revisarlo ya que según la documentación de AWS las funciones lamdas ya son gestionadas por el propio proveedor.

Mirando más por la seguridad por el propio despliegue de la arquitectura hemos querido tener dos partes muy diferenciadas. La parte de lectura o la que va a ver los usuarios, ya que no se ha considerado que el usuario pueda realizar modificaciones vía web de los registros de la base de datos. De esta forma podemos evitar que accesos no deseados a través de las instancias web pues estarán únicamente comunicadas con las instancias de solo lectura de base de datos. Y por otro lado la parte de las Raspberry's o escritura que se ocupará del volcado de datos y que gracias a las RestAPI se podría determinar y controlar desde que IP se puede escribir o no, etc. Por último, también se ha pensado en utilizar una instancia de monitoreo de la actividad de la aplicación con un CloudWatch, aunque actualmente no tenemos constancia si podemos utilizarlo con el plan académico de AWS.

En cuanto a la disponibilidad desde nuestro punto de vista y como ha sido diseñada la aplicación, consideramos que por el tipo, la utilidad y la criticidad de la misma, la mejor manera de asegurar la disponibilidad de la misma es utilizar el método de Backup y Restore visto en la asignatura primero por gasto y segundo porque ya la propia de la arquitectura está pensada con este método en mente. El resto es cierto que aseguran mayor disponibilidad, aunque a nuestro juicio el incremento de coste y complejidad no los llegan a justificar.

Por otra parte, también debemos de asegurar evitar puntos únicos de ruptura que puedan provocar un fallo en la arquitectura. Desde el punto de vista del diseño presentado hemos detectado que un punto de conflicto puede llegar a ser la RestAPI que si no se lleva a cabo una correcta configuración y diseño de esta puede llegar a ser el punto crítico de la aplicación, provocando que un posible atacante acceda, introduzca información errónea o incluso que provoque la caída de la RestAPI y que la aplicación no reciba la información de la Raspberry. Para ello, se ha pensado en realizar un control de seguridad para controlar quien tiene acceso y posibilidad de realizar consultas a dicha API, bien sea a través de la IP pública de la red en la que se encuentre la Raspberry, facilitando si se sitúan más raspberry en la misma red. U otra posibilidad es a través de MAC, lo que provocaría introducir a mano las MAC de posibles nuevas Raspberry en caso del escalado de esta parte de la arquitectura.

Pensamos que con la arquitectura que proponemos, tenemos una aplicación que puede suplir las necesidades del usuario tanto a nivel de funcionalidad como de escalado si se diera la necesidad de generar más instancias o duplicar los recursos inicialmente pensados.