



# PROYECTO ABP VER.0.1 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.

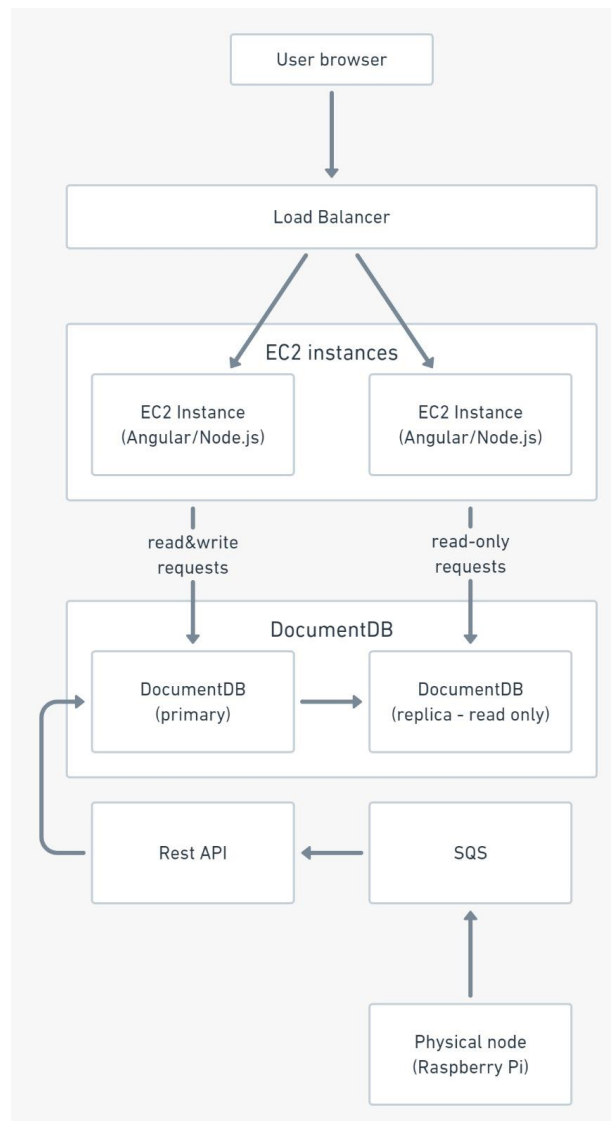


Figura 1 Estructura aplicación Big-Eye

A continuación, se van a definir y detallar cada uno de los 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 ha 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 accede (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 pérdida 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:** Se utilizarán instancias EC2 con un ligero BackEnd basado en Node.js para establecer la comunicación entre la RaspberryPi y la base de datos, evitando establecer una conexión directa con la base de datos y así prevenir posibles fallos de seguridad o intrusiones no deseadas a los datos.
- **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 establece 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.

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.

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.