The use case

Description of the business use case selected

Taxify es una empresa que tiene más de 500 taxistas autónomos que transportan paquetes y personas de forma que lo compaginan con su vida. Cada conductor hace múltiples viajes al año transportando personas o paquetes de distintas características de modo de buscan coordinar sus viajes usuales con servicios de transporte para economizar sus viajes (estilo blablacar).

El pago de los servicios funciona de tal forma que no pueda haber ni explotación del trabajador, ni aprovechamiento económico por parte de los conductores más allá de ayudar a cubrir sus gastos. El sistema de pagos evita el funciomaniento actual capitalista de la oferta-demanda y busca la igualdad entre los usuarios. Para tal efecto, debido a que los vehículos no estan equipados con un taxímetro, la aplicación funciona a través de un sistema de reconocimiento de los valores de kilometraje y consumo a partir de fotografias de los cuadros de mando. Al iniciar y terminar el servicio, el conductor hace unas fotos de su cuadro de mando que adjunta a la tarea para que le sean analizados los gastos de ese trayecto y se pueda hacer el cálculo del gasto de ese trayecto.

La misma aplicación también tiene la opción de usarse para uso personal. Así tiene las funcionalidades de devolver un análisis exhaustivo de los gastos del coche reportando un análisis de los datos en un gráfico o funcionalidades como las de dividir los gastos de un viaje entre los amigos.

Description of the technical requirements or assumptions done

Se requiere:

♦ Un lugar donde la información y par de fotos de cada viaje sea almacenada de forma lógica y ordenada, construida de tal forma que se pueda acceder más tarde a la información con un programa y que esté guardada en un entorno seguro. O sea, se requiere una **base de datos**.

La base de datos tendrá que organizar datos no estructurados como son las imágenes y los demás datos característicos de la carga (paquetes y personas) que transporta será de forma estructurada.

Se requieren distintas bases de datos. Una con la información de cada viaje enviado (semiestructurado), una con la información extraída de la imagen ya en forma totalmente estructurada y otra con la información relacionada entre si para ser representada en reportes y gráficos.

• Un sistema que defina las relaciones entre la base de datos, determine condiciones, normas y límites en la base de datos. O sea, un **schema**.

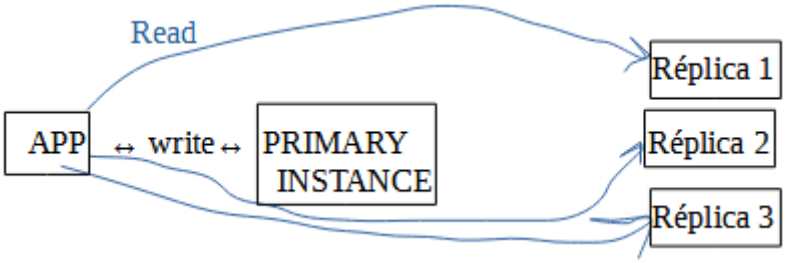
• Una compilación que permita una **consistencia** entre los datos, que asegure devolver datos válidos y que asegure que no se hacen cambios en las filas de datos hasta que todos los datos en todas las réplicas se han realizado correctamente.

• Una compilación que permita una **atomicidad** entre los datos, que asegure que las transaciones se realizan o fallan completamente para volver a realizarse de nuevo. Asegurando que no haya inserciones de datos erróneos.

• Una compilación que permita una **isolación** entre los datos, que asegure que un proceso de datos no interfiera ni se mezcle con otro proceso de datos.

• Una compilación que permita una **durabilidad** entre los datos, que permita que los cambios en la base de datos se mantienen.

• En definitiva, se requiere **compilance tipo ACID** (Atomicity, Consistency, Isolation and Durability) que no se permita hacer modificaciones de la base de datos hasta que todas las réplicas se hayan actualizado correctamente. A su vez que permita devolver la versión más reciente de los datos para que ese sea procesado lo más rápido posible los nuevos datos introducidos para poder dotar al usuario de un análisis rápido. En consecuencia, se requiere una

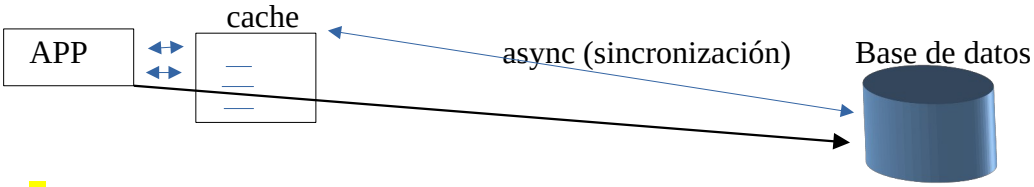


Sistema database ACID que permite la lectura directa y que tiene una escritura controlada por consistencia.

• Un **método OLTP** (Online Transaction Processing) que focaliza su atención en actualizar, inserir y eliminar las transaciones de datos. Porqué en nuestra aplicación tenemos consultas simples y cortas que requieren poco espacio y tiempo para ser procesadas, como son las consultas de cuánto me ha costado este viaje usando una transacción a partir de un par de fotos y poca información complementaria.

• Una **Database Management System** (DBMS) que permita a los usuarios y aplicaciones interaccionar con los datos.

• Acceder a los datos del rendimiento del usuario a través de la aplicación de forma casi instantánea. O sea se requiere que los datos sean cargados muy rápidamente (*cache*) sin necesidad de hacer una consulta que tarde más tiempo. Por lo tanto necesitamos una **base de datos "in memory"** como por ejemplo Amazon ElastiCache. Normalmente esta está conectada a otra base de datos.



♦ Un lugar donde verter toda información en distintos formatos: estructurado y no estructurado (como las imágenes). O sea, se necesita un **DataLake.** Ese funciona a través del método OLAP (Online Analysis Processing) que permite almacenar datos históricos que han sido introducidos a través de OLTP. En el datalake no se requiere un esquema para definir la ingesta y puede que no esté actualizado al instante. El datalake simplifica la ingesta de datos directos del usuario y permite separar el almacenamiento de la computación.

♦ Un lugar donde guardar la información altamente estrucutrada que ha sido preprocesada después de su recolección y antes de su análisi. O sea, un **Data Warehouse**. En este caso se requiere que solo almacene datos de un sistema o fuente y el almacenaje sea muy optimizado para el análisi.

Durante el proceso de ETL (Extract, Transform and Load data) de la base de datos al Data Warehouse es probable que el Data Warehouse no tenga la versión más actualizada de los sistemas. En nuestro caso, se requiere que lo tenga lo más pronto posible para que el usuario tenga el análisis en pocos minutos. Eso sería fácil de lograr por el hecho que solo hay un sistema de donde provengan los datos y el tiempo de computación no es muy elevado para las tareas a realizar. Se requiere ir con cuidado en este aspecto ya que los data warehouses no están construidos con la intención de satisfacer las necesidades de que una aplicación tiene que ser simultánea y transaccional.

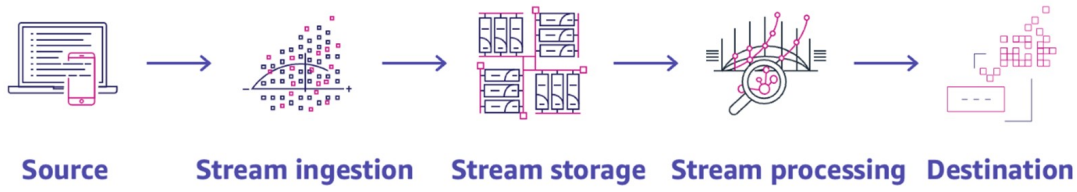
♦ Un lugar donde realizar el **Análisi de Datos** a través de herramientas de **BI** (Business Inteligence). Los Business analysts se connectan a los Data Warehouses a través de Herramientas BI para analizar los datos y sacar KPI (Key Points Indicators) y gráficas que son mostradas en la aplicación.

♦ Una secuencia de pasos de procesado que mueva datos de un lugar a otro (Data Source a Data Storage, Data Lake a Data Warehouse, Data Warehouse a Data Analytics, Data Storage a Analysis, Data Analytics a Graficar, Data Base a Cache, Cache a Aplicación, etc.). O sea, requerimos de **Data Pipelines**. Estos se usan para ingestar, transformar y entregar datos por distintos procesos como son el análisi, el almacenaje o la integración. Durante el proceso se necesita que los datos pasen por algunos pasos de procesamiento.

• Un tipo de Data Pipeline en **streaming** ya que se requiere una constante carga de datos, actualización y análisis de esos en tiempo real. Cada acción, como un viaje realizado, se considera un evento y cada evento relacionado, como finalización de viaje por parte del cliente o finalización de la entrega del paquete se necesita que esté agrupado junto en un mismo tema o *stream*. Estos eventos se tendrán que procesar rápidamente después de que ocurran por lo que se requiere una más baja latencia que los procesos batch.

• Asumir y considerar un tipo de Data Pipeline en **batch** donde la carga de los datos se realizan al final del día, cuando son analizados y devueltos los resultados en caso que se considere que no hay una necesidad imediata que analizar los viajes. Pero no sería lo más deseable para este proyecto ya que los usuarios y clientes quieren tener la respuesta immediata. Por lo tanto se considera que no se necesita este tipo de requerimiento técnico. En cambio:

• Una **arquitectura moderna de streaming** que permita una ingesta, procesado y análisi de altos volúmenes de datos a una gran velocidad y en tiempo real para construir una experiencia para el cliente más atractiva tal que así:



♦ Transformar, organizar, procesar, graficar, explicar el resultado y sacar conclusiones de los datos. O sea, se requerimos de **Data Analytics** y **Data Visualization**. Para acabar mostrando el resultado de los análisis descriptivos del consumo a cada uno de los usuarios así como un análisi predictivo de lo que va a gastar en el futuro.

The tools

Research on the available market tools

Para cada uno los requerimientos técnicos necesitados existen las siguientes herramientas:

♦ Sobre las Data Bases:

Amazon Relational Database Service (Amazon RDS)

Amazon DynamoDB

Amazon Neptune

Amazon Aurora

Amazon DocumentDB (with MongoDB compatibility)

Amazon ElastiCache

♦ Sobre los Data Lakes:

Amazon Simple Storage Service (Amazon S3)

AWS Lambda

Amazon Kinesis

AWS DataSync

AWS Lake Formation

Según si es más fácil el aceso a los datos (←) o si hay mayor duración del almacenaje (→) tenemos:

S3 Standard

S3 Intelligent-Tiering

S3 Standard-IA

S3 One Zone-IA

S3 Glacier Flexible Retrieval

S3 Glacier Deep Archive

S3 Glacier Instant Retrieval

S3 on Outposts

♦ Sobre el Data Warehouses:

Amazon Redshift

♦ Sobre los Aplicativos:

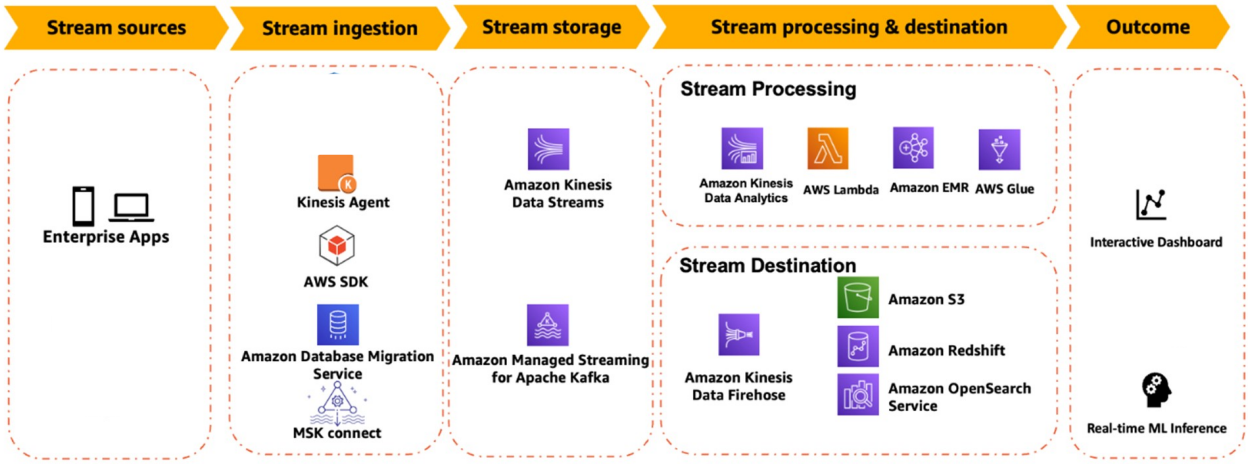
♦ Sobre los Data Pipelines:

Amazon EMR (Streaming)

AWS Step Functions

AWS Glue workflows

♦ Sobre el pipeline de la Arquitectura en Streaming se consideran las siguientes herramientas:



En concreto:

Amazon Kinesis

Amazon Managed Streaming for Apache Kafka (Amazon MSK)

Amazon EMR

Amazon Kinesis Data Streams

Collect and store data streams for analytics

Amazon Kinesis Data Firehose

Load data streams onto AWS data stores and third-party destinations

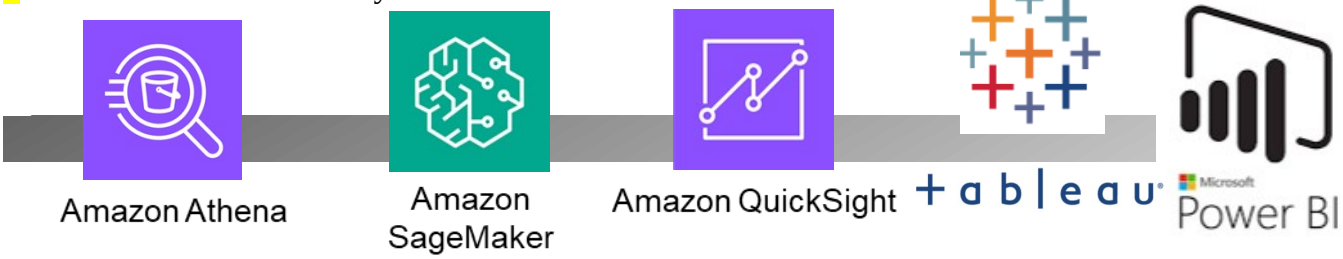
Amazon Kinesis Video Streams

Collect and store video streams for analytics

Amazon Managed Service for Apache Flink (successor to **Kinesis Data Analytics**)

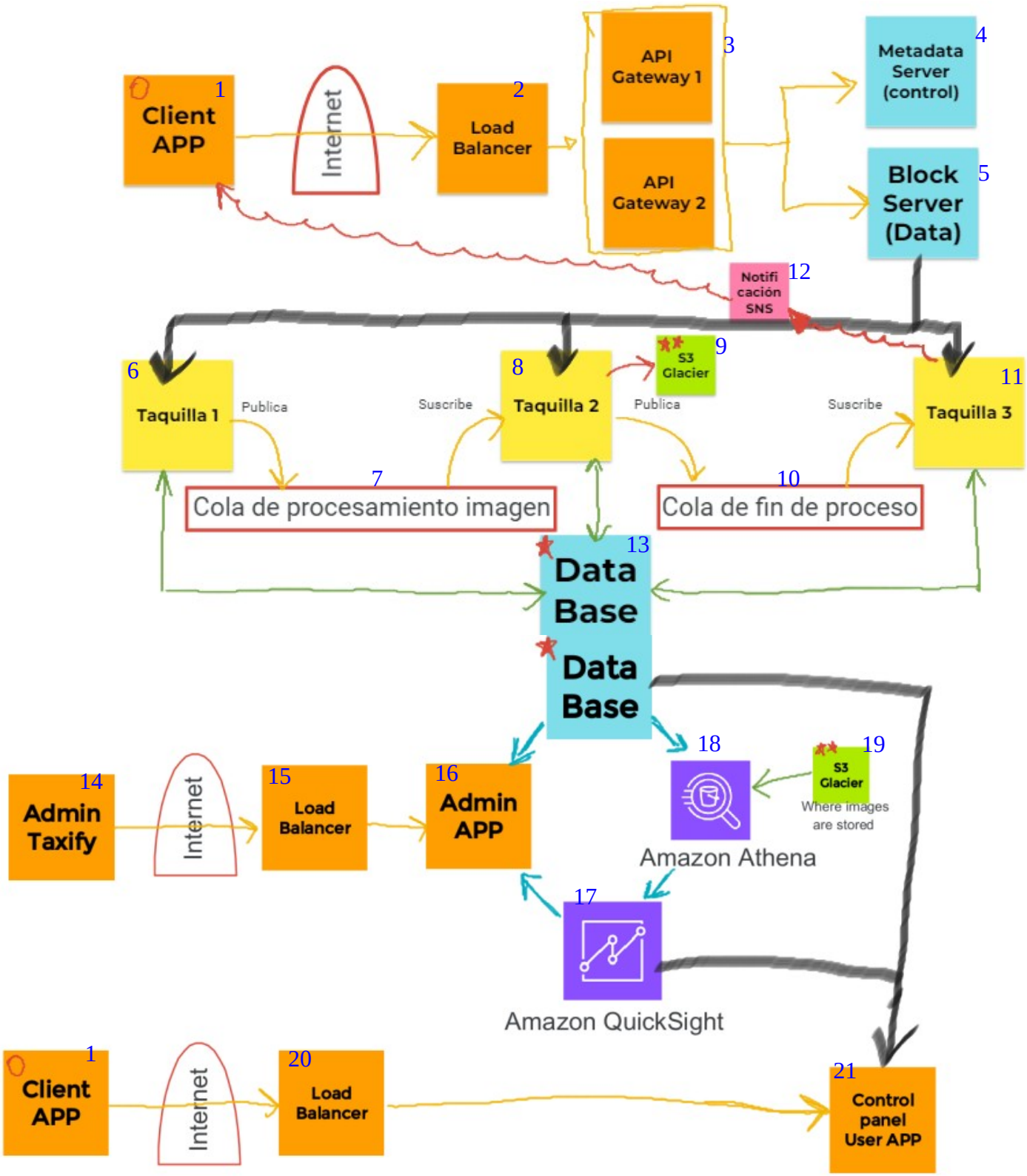
Analyze data streams with Kinesis Data Analytics Studio or Apache Flink

♦ Sobre el Análisis de Datos y la Visualización de Datos:



Proposed problem solution

General architecture diagram



Explicación del diagrama de estructura superior

• Cliente (1): es el usuario que toma las fotografías y, a través de la app con internet las manda.

• Balanceador de cargas (*Load balancer*) (2): Componente de hardware o software que se encarga de recibir peticiones internas y distribuirlas para que se mantenga la calidad del servicio en las conexiones. Distribuye las peticiones de los clientes en la entrada para que no se sobrecarge la puerta de entrada de datos.

• API Gateway (3): Componente en el que se registran peticiones en paral·lelo para eliminar posteriores operaciones innecesarias del aplicativo. Actúa como capa de seguridad de control del dato. Valida el cliente, sus credenciales y si es un cliente activo o no. Bloquea las sobrepeticiones para evitar sobrecargar el sistema. Por ejemplo bloquea peticiones en el caso que haya muchas peticiones por usuario o más peticiones que las que el sistema puede soportar.

• Entrada al sistema:

- Si se trata de una prueba que hemos realizado para comprovar el control, va hacia la rama de control. (4)

- Si se trata de dato de entrada va al bloque de servidor. (5)

• Block server y taquillas (5 a 13):

El servidor está formado por 3 taquillas de procesamiento (6, 8, 11). El servidor monta un mensaje lógico con la información que ha llegado y lo envía a la taquilla de entrada, taquilla 1 (6). Esta taquilla por un lado guarda los datos identificador e información del viaje en la base de datos (13). Por otro lado publica un mensaje en la cola de procesamiento de la imagen (7) con orden FIFO (*First In First Out*) que será una tarea a la que se suscribirá la taquilla 2 (8) . La taquilla 2 es un aplicativo que le llegará la tarea con el par de imágenes a analizar. Leerá la tarea y realizará los procesamientos de imagen necesarios hasta sacar las métricas de kilometraje y consumo de las imágenes. El proceso se realizará seguramente con distintos trabajadores (*workers*) en paral·lelo elásticamente. La taquilla 2 acude a la base de datos para añadir dicha información junto con la información del viaje (13) y guardará las imágenes en el data lake S3 (9) glacier con el mismo identificador de tarea. En terminar la taquilla 2 la tarea, publicará el mensaje de finalización de proceso a la cola correspondiente (10). La taquilla 3 (11) está suscrita a dicha cola y en recibir el mensaje de que se ha terminado el proceso, lo mandará via notificación SNS (12) a la aplicación del cliente. La notificación funciona con un sistema DNS (Sistema de Nombres de Dominio) que dice en qué direccion corresponde con qué nombre de dominio. Para que el cliente (1) acabe recibiendo los datos que le tocan. El proceso que realiza la taquilla 3 es coger los resultados del viaje y realizar el proceso de cálculo de tarificación, pago y cobro datos guardados en la base dde datos (13).

Explicación del diagrama de estructura inferior

El administrador de Taxify (14) ingresa en la aplicación de administración siempre a través de internet y de un balanceador de carga (15). Allí tiene aceso a los datos de la base de datos (13) y a los gráficos de QuickSight (17). Para la creación de los gráficos, Amazon Athena (18) lee los datos y procesa las analíticas. La información analizada por Athena se carga en QuickSight y finalmente QuickSight carga los *dashboards*.

De forma paralela, el cliente (1) que ingresa a través de la aplicación tiene acceso al control de panel del usuario (21) desde donde puede consultar sus datos y gráficos.

Decided tools and justification on why they were chosen. Focus as much as possible to AWS Services.

♦ Como **Data Bases** se consideran como posibles Aurora y RDS (Relational Database Service). La diferencia entre ellas está en la arquitectura. Aurora tiene una arquitectura distribuida con el objetivo de aumentar la disponibilidad. RDS en cambio tiene una arquitectura no distribuida, sino con réplica. Entre las dos, en el proyecto nos decantamos por Aurora ya que es la que ofrece mayor disponibilidad. Como puntos positivos Aurora es la más barata y como desventaja es que no tiene la última versión de la base de datos.

Fuentes: https://aws.amazon.com/es/rds/aurora/ , https://aws.amazon.com/es/rds/?p=ft&c=db&z=3

Escogido

♦ Como **Aplicativos** para desplegar y escalar “taquilla 1”, “taquilla 2”, “taquilla 3” y admin se escoge usar el contenedor para aplicativo AWS Elastic Beanstalk. También podríamos usar EC2 (Amazon Elastic Compute Cloud) más un administrador de sistema. Pero Beanstalk será la mejor ayuda ya que nos eliminará las tareas de administración de las instancias. Además escala automáticamente el aplicativo regulando el número trabajadores en paralelo, dos características que necesitaremos en nuestro proyecto.

Fuentes: https://aws.amazon.com/es/elasticbeanstalk/ , https://aws.amazon.com/es/ec2/

Escogido

♦ Como **Balancedor de Carga** para gestionar el tráfico usaremos Elastic Load Balancing (ELB).

Fuente: https://aws.amazon.com/es/elasticloadbalancing/

Pensamos que no vamos a necesitar cloudfront como CDN ya que no vamos a necesidad cargar grandes cantidades de imágenes o formatos en la aplicación. VPS y VPC tampoco se van a necesitar ya que vamos a usar directamente los recursos.

♦ Como **API Gatway** usaremos el único disponible en AWS.

Fuente: https://aws.amazon.com/es/api-gateway/

♦ Para el **Análisis de Datos** y la **Visualización de Datos** vemos que necesitamos Athena y QuickSight:

Fuentes: https://aws.amazon.com/athena/ , https://aws.amazon.com/quicksight/

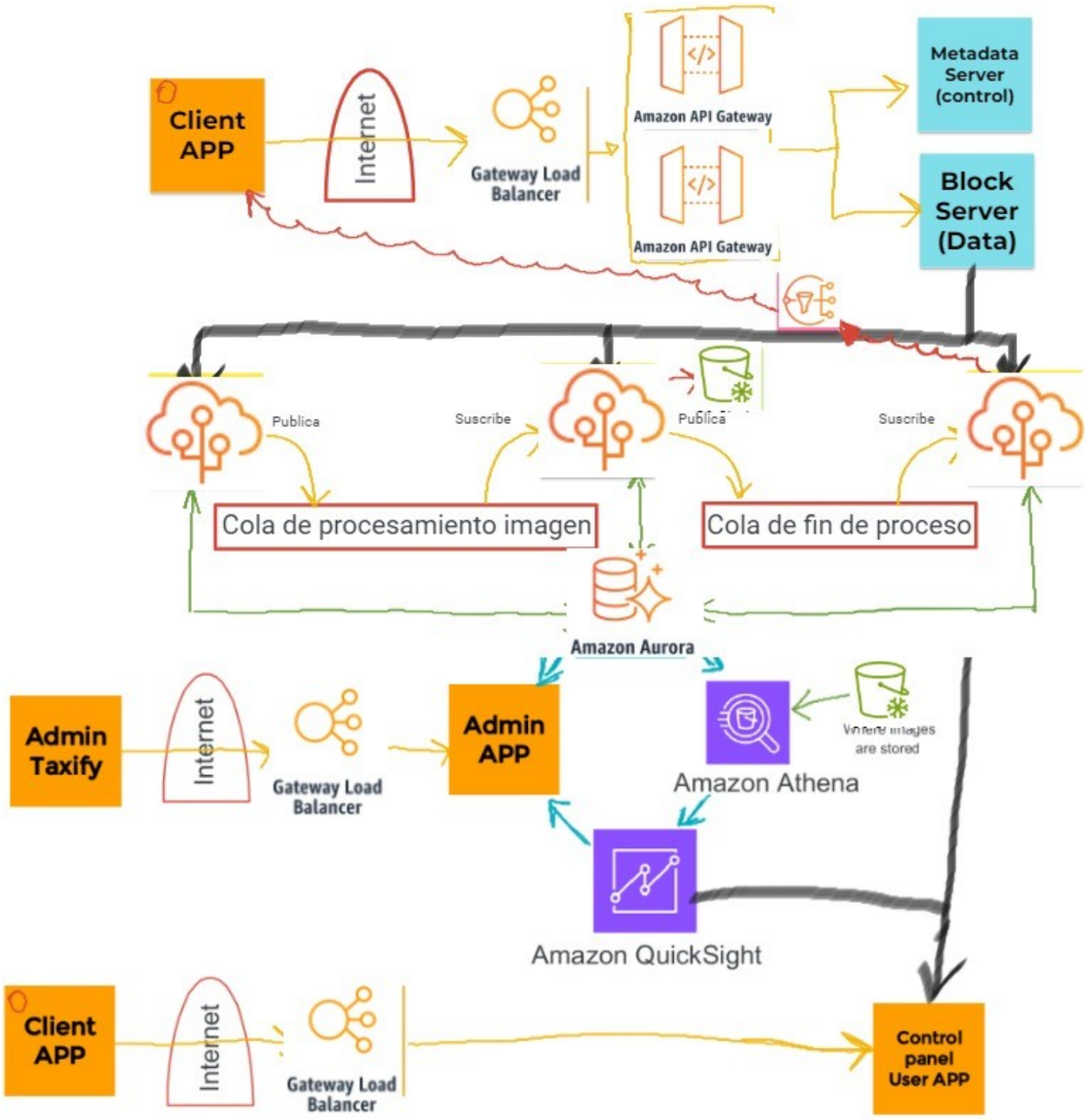
♦ Como **Data Lake** usamos S3 Glacier debido a su respuesta en milisegundos, escalabilidad ilimitada, durabilidad de 11 nueves, seguridad y coste bajo.

Fuente: https://aws.amazon.com/es/pm/s3-glacier/

♦ Como **Notificador SNS** que la taquilla 3 utilizará para notificar al usuario usaremos Amazon Simple Notification Service.

Fuente: https://aws.amazon.com/es/sns/

En consecuencia a las herramientas escogidas, la arquitectura quedaría así:



Optional) General explanation of the code and how to test it

Cost Analysis

♦ Amazon aurora → https://calculator.aws/#/createCalculator/AuroraMySQL

Precio variante según región, tipo de instancia, capacidad de almacenaje y caracterísitcas adicionales.

Considerando las siguientes características, el precio sería de: 30.546,12 USD/año

Región	España
Tipo	Compatible con MySQL de Aurora
Nodos	1 db.r6g.large

♦ AWS Elastic Beanstalk → https://aws.amazon.com/es/contact-us/sales-support/?pg=elasticbsprice&cta=herobtn

Precio variante según región, tipo de instancia, almacenamiento, trafigo de red y otros recursos asociados a la aplicación.

Hay que pedir presupuesto indicando registrando usuario y empresa.

♦ Elastic Load Balancing (ELB) → https://calculator.aws/#/createCalculator/shield

Precio es variante según si es de tipo *network loader balancer* o es de tipo *application loader balancer* y según la región, cantidad de tráfico procesado y otros servicios asociados.

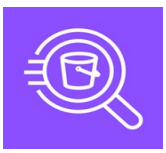
Considerando la región de España se estima un costo de 72.000 USD/año.

♦ Amazon API Gateway → https://calculator.aws/#/createCalculator/APIGateway

Precio es variante según varios factores como la región, el número de solicitudes, tipo de API Gateway (REST API o WebSocket API), tamaño de la carga útil (Payload Size) y ancho de banda.

Considerando la región de España con 2000 solicitudes al mes (4 por usuario de media) se estima un costo de 6.003,23 USD/mes.

♦ Athena → https://calculator.aws/#/createCalculator/Athena



El precio es variante según factores como la región, la cantidad de consultas escaneadas y ejecutadas, el tiempo de ejecución de consultas, el almacenaje y transferencia de datos.

Considerando España con 2000 consultas por día de 10GB el precio es de 3.356,51 USD/mes.

♦ QuickSight → https://calculator.aws/#/createCalculator/QuickSight

El precio es variante según la región, número de usuarios activos, capacidad SPICE, número de conjuntos de datos, transferencia de datos y ancho de banda.

Considerando España con 2000 consultas por día con 22 días laborables, 5% de lectores activos dos veces por dia, 50% de lectores frecuentes dos veces por semana, 25% de lectores ocasionales dos veces por mes y 20% de lectores inactivos almenos una vez al mes, el precio es de 3.218,00 USD/mes.

♦ S3 Glacier → https://calculator.aws/#/createCalculator/S3

Primero hay que escoger el tipo de Data Lake S3. En nuestro caso el S3 Glacier Flexible Retrival. Entonces el precio es variante según factores como la región, volumen de almacenamiento, cantidad de transferencia de datos o tarifas de recuperación de datos.

Considerando España, 2000 GB de capacidad total al mes (1MB/par de imágenes * 2000 imágenes al mes/usuario * 500 usuarios = 977 GB + otras informaciones extras), tamaño medio de objeto de 500 KB sale un precio de 9,36 USD/mes.

♦ Amazon Simple Notification Service → https://calculator.aws/#/createCalculator/SNS

Su precio varía según la cantidad de mensajes publicados enviados a través de distintos medios ya sea SMS, push del móvil, suscripciones de entrega de mensajes y tarifas de entrega de mensajes, así como la región de AWS.

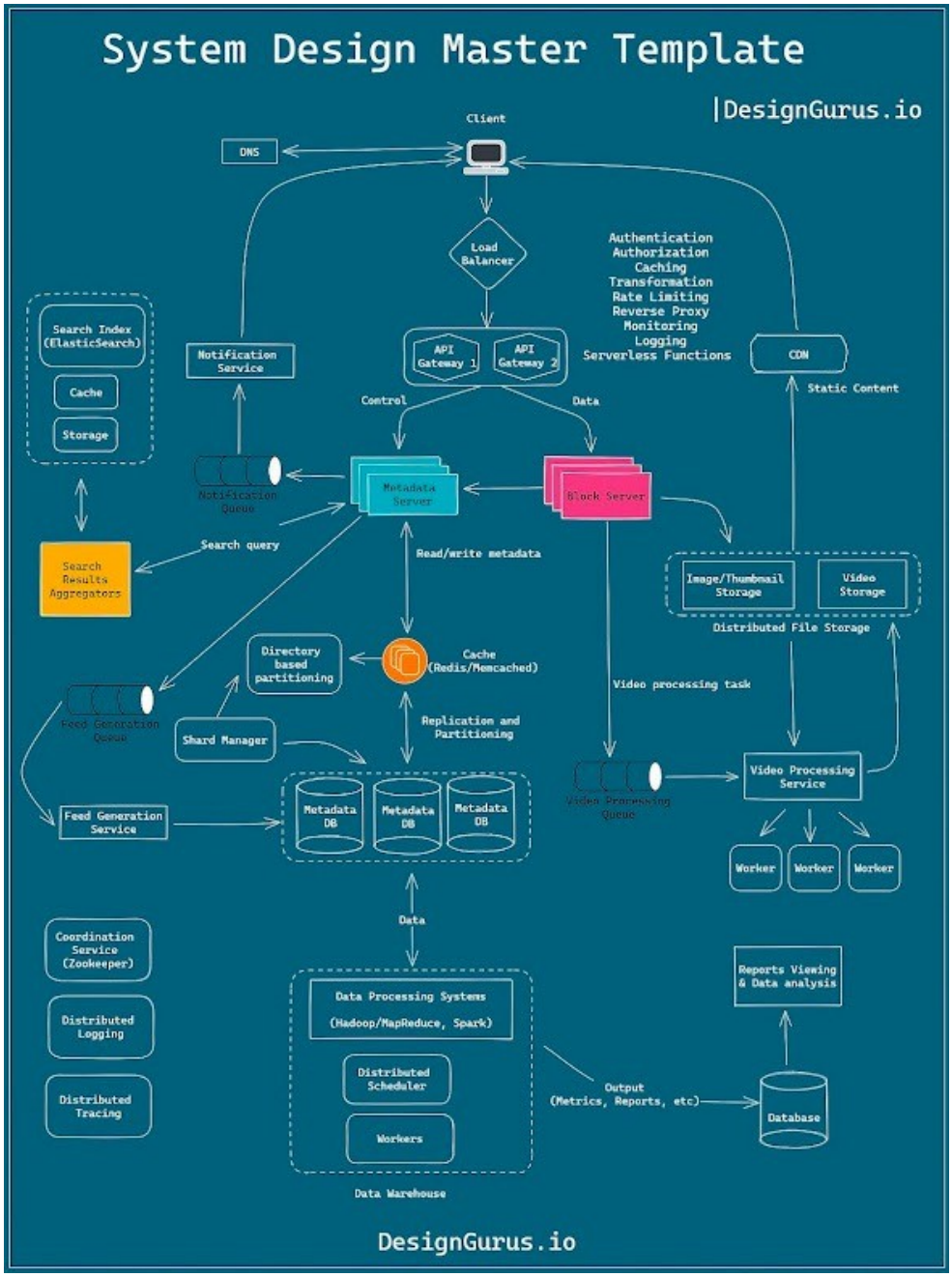
En España 2 millones de solicitudes al mes tienen un precio de 0,50 USD.

Next steps and potential improvements

Poner a prueba la arquitectura y ver que punto de la arquitectura va lenta para poder remplazarla por un producto que vaya más rápido. Un punto crítico puede ser la rapidez de la taquilla 2 en analizar las imágenes ya que es un punto que suele tardar bastante tiempo y ralentizar toda la cadena de proceso. Las mejoras en los gráficos también pueden aportar una calidad sustancial en la aplicación. Por último, hay que considerar el escalado del sistema en caso de que la cantidad de clientes y entradas de datos aumente. Habrá que tener en cuenta los sistemas de seguridad y filtro de mensajes cuando estos empiezen a ser grandes.

Annexo

Sistema general de carga de datos y salida de datos en el que se ha apoyado el proyecto:



CDN. Content Delibery Network. Capa de caché de imágenes, vídeo o audio que muestra la página web. Está en caché para evitar usar recursos del servidor par cargar siempre las mismas imágenes y tipografia de letra necesario para mostrar la página web o aplicación móbil. → no es necesario para nuestra aplicación ya que la app no requiere cargar muchos elementos.