

**S.Ma.R.T. (Swim Mapping, Reporting & Training)**

**TRABAJO FIN DE MÁSTER**

Programa: **Máster Executive en Big Data, Cloud & Analytics**

Período

académico: **2019 – 2020**

Autores: **Alfonsel Jaén, Carlos**

**Mendivil García, Álvaro**

**Oliva Sanz, Alberto**

**Phares Orejana, David**

**Trabajo Fin de Máster**

**Título**: S.Ma.R.T. (Swim Mapping, Reporting & Testing)

**Autores**: Alfonsel Jaén, Carlos

Mendivil García, Álvaro

Oliva Sanz, Alberto

Phares Orejana, David

**Tutor**: Arias Martínez, Jacinto

**Miembros del Tribunal**

**Presidente**: ………………………………………………

**Vocal**: ………………………………………………

**Secretario**: ………………………………………………

**Suplente**: ………………………………………………

**Fecha de lectura**: Madrid, ……. de ……………………… de 2020

**Calificación**: ………………………………………………

**Resumen**

**(PENDIENTE)**

**Palabras clave**

**(PENDIENTE)**

**Abstract**

**(PENDING)**

**Keywords**

**(PENDING)**

**Agradecimientos**

**Contenido**

[1 Introducción 8](#_Toc52703259)

[1.1 Contexto y justificación del proyecto 8](#_Toc52703260)

[1.2 Objetivos 9](#_Toc52703261)

[1.3 Enfoque y método seguido 10](#_Toc52703262)

[1.4 Planificación del proyecto 10](#_Toc52703263)

[2 Arquitectura 11](#_Toc52703264)

[2.1 Arquitectura actual 11](#_Toc52703265)

[2.2 Arquitectura propuesta 12](#_Toc52703266)

[3 Ingesta y almacenamiento del Dato 16](#_Toc52703267)

[3.1 Razones para automatizar la ingesta de datos 16](#_Toc52703268)

[3.2 Edge node 17](#_Toc52703269)

[3.2.1 Configuración 18](#_Toc52703270)

[3.2.2 Componentes 19](#_Toc52703271)

[3.2.3 Integración on-premise 22](#_Toc52703272)

[3.3 Simulador piscina 23](#_Toc52703273)

[3.4 Amazon S3 26](#_Toc52703274)

[3.4.1 Creación del bucket 27](#_Toc52703275)

[3.4.2 Estructura de carpetas 30](#_Toc52703276)

[3.4.3 Coste 30](#_Toc52703277)

[3.5 Kinesis data stream 31](#_Toc52703278)

[3.5.1 Creación y configuración 32](#_Toc52703279)

[3.5.2 Coste 33](#_Toc52703280)

[3.6 Kinesis Data Firehose 34](#_Toc52703281)

[3.6.1 Configuración 34](#_Toc52703282)

[3.6.2 Configuración de accesos 37](#_Toc52703283)

[3.6.3 Coste 37](#_Toc52703284)

[4 Limpieza y Preparación del Dato 38](#_Toc52703285)

[5 Analítica y Modelo de Aprendizaje 39](#_Toc52703286)

[6 Servicio de información y visualización 40](#_Toc52703287)

[7 Propuestas de mejora 41](#_Toc52703288)

[8 Resumen Final y Conclusiones 42](#_Toc52703289)

[9 Código Fuente 43](#_Toc52703290)

[9.1 connectivity.py 43](#_Toc52703291)

[9.2 serverqueue.py 43](#_Toc52703292)

[9.3 piscina.py 43](#_Toc52703293)

[9.4 nadador.py 43](#_Toc52703294)

[9.5 SMART\_Gestión\_Piscina.ipynb 43](#_Toc52703295)

[10 Bibliografía 44](#_Toc52703296)

# Introducción

## Contexto y justificación del proyecto

Este proyecto surge con la necesidad de evolucionar el sistema que actualmente se encuentra en funcionamiento en las piscinas gestionadas por nuestro partner [**NBN23**](https://www.nbn23.com/es/) hacia un entorno Cloud que permita ampliar las posibilidades que dicho sistema ya ofrece hoy en día.

Hemos partido del concepto de las 7 V’s ([*Infografía Big Data: las 7 V*](#_Bibliografía)) para el análisis de la situación:

* **Volumen.** **Cantidad de datos** que son generados y se almacenan con la finalidad de procesarlos para transformar los datos en acciones.
* **Velocidad. Velocidad** es la rapidez en la que los datos son creados, almacenados y procesados en tiempo real.
* **Variedad. Formas, tipos y fuentes** en las que se registran los datos: documentos de texto, correos electrónicos, audios, vídeos o imágenes que tenemos en nuestro dispositivo móvil, perfiles de redes sociales, etc.
* **Veracidad.** Es la **incertidumbre de los datos**, es decir, el grado de fiabilidad de la información recibida.
* **Viabilidad.** Capacidad que tienen las compañías en generar un **uso eficaz del gran volumen de datos** que manejan.
* **Visualización. Modo en el que los datos son presentados** para encontrar patrones y claves ocultas en el tema a investigar.
* **Valor. Datos que se transforman en la información**; ésta a su vez se convierte en conocimiento y éste en acción o en decisión.

El entorno en el que nos vamos a centrar es en el de las piscinas cubiertas, donde los usuarios de la aplicación (los nadadores), mediante un tag enganchado al gorro de natación o a las gafas, van a poder ver cómo ciertos parámetros van reflejándose en tiempo real en un monitor de visualización, además de obtener una serie de estadísticas de performance al final de la sesión.

## Objetivos

Una de las limitaciones de los usuarios es la colocación del tag en una zona del cuerpo que no permanezca sumergida un tiempo excesivo, ya que la comunicación entre dichos tags y los sensores colocados alrededor de la piscina es a través de un protocolo radio (Bluetooth) que no funciona bajo el agua. Esta limitación inicial se transforma en una característica a explotar ya que nos va a permitir detectar, entre otros, posibles ahogamientos: si un tag deja de ser registrado durante X segundos, una alarma visual aparecerá en el monitor indicando que algo puede estar sucediendo en la piscina y un aviso será enviado a los relojes de los socorristas para centrar su atención en el punto de la piscina correspondiente.

El título de nuestro TFM viene a identificar los objetivos que buscamos implementar con nuestro proyecto: **S**wim, **MA**pping, **R**eporting & **T**raining (SMART). Es decir, que nuestro proyecto persigue crear una arquitectura que permita la ingesta de datos en tiempo real desde los sensores, así como el tratamiento intermedio del dato para poder hacer una explotación final del mismo permitiendo así aportar al centro y a los usuarios información que les permita estar al tanto de su situación, así como posibilidades de mejora en distintos ámbitos.

## Enfoque y método seguido

(PENDIENTE)

## Planificación del proyecto

(PENDIENTE)

# Arquitectura

## Arquitectura actual

El sistema que se encuentra actualmente en funcionamiento e implementado en los centros consta de una serie de sensores (antenas de radiofrecuencia) situados alrededor de la piscina. Estos sensores son los encargados de recibir las señales enviadas por los dispositivos bluetooth que llevan los nadadores sujetos en sus gorros o gafas y transmitir esta información directamente a una base de datos MongoDB donde es almacenada.



Existe una serie de aplicaciones que hacen uso de la información almacenada en este sistema para proporcionar distintos servicios:

* Panel situado en la piscina donde se muestra la información de cada una de las calles (distancia, nadadores, etc.).
* Aplicación online para la consulta de resultados de cada entrenamiento.

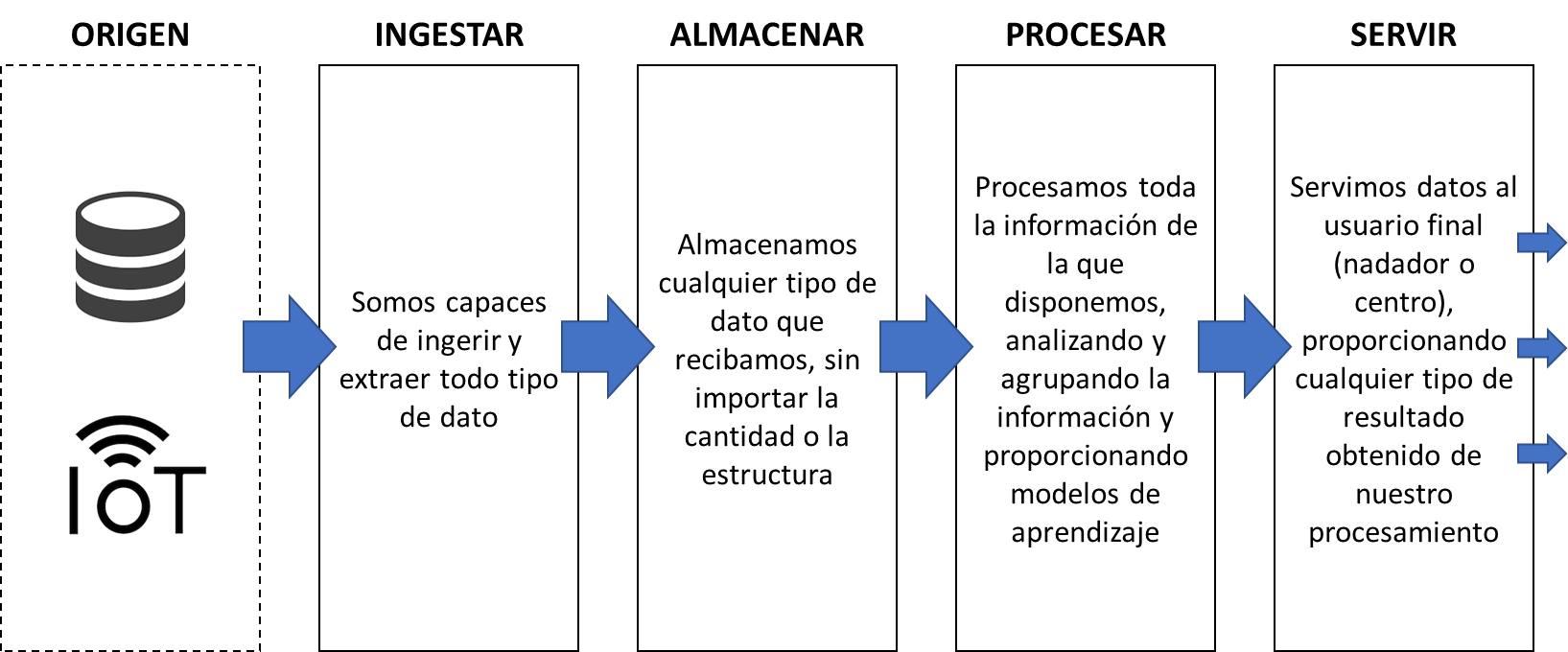
## Arquitectura propuesta

Para esta primera versión de nuestra arquitectura vamos a proponer trabajar en un entorno híbrido. El motivo de esta decisión es la de interferir en lo mínimo sobre la operativa actual que las piscinas de nuestro partner están siguiendo a nivel técnico. Más adelante propondremos alternativas al modelo actual, aportando arquitecturas que sean más escalables y permitan una gestión mayoritariamente en la nube.

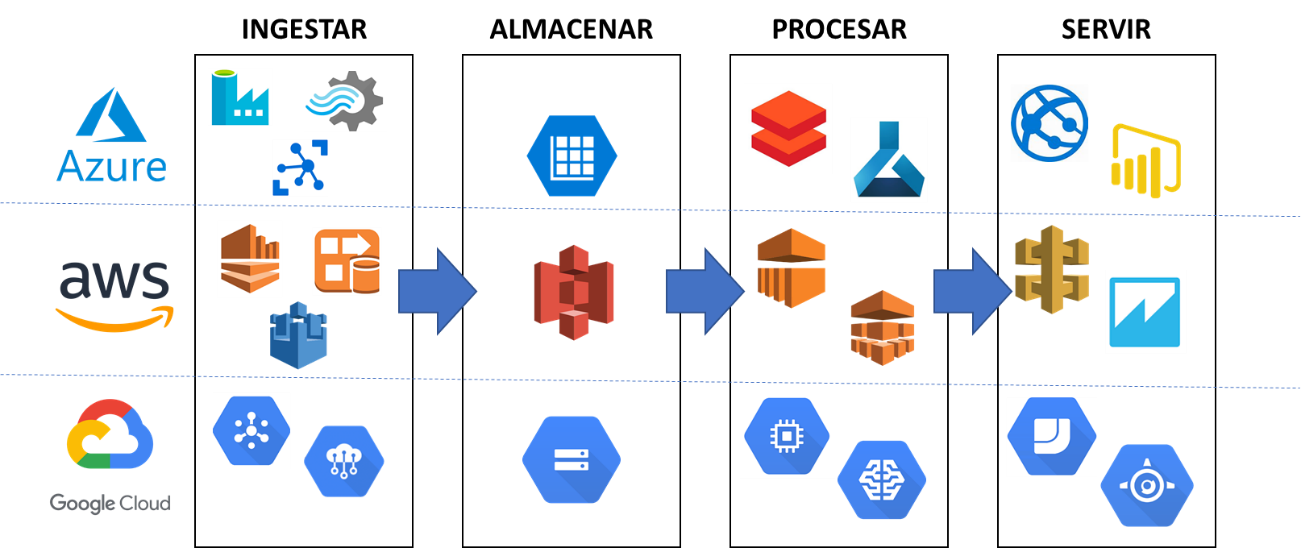
En el entorno actual, como ya se indicó en el apartado anterior, toda la información enviada por bluetooth desde los distintos dispositivos que llevan los nadadores se recibe en unos sensores colocados alrededor de la piscina. Estos sensores registrarán esta información de forma local en los servidores de la propia piscina en una instancia de MongoDB preparada para dicho propósito.

La arquitectura que proponemos contempla todas las etapas necesarias para hacer llegar esa información registrada localmente a la nube, con el objetivo no sólo de almacenarla sino de procesarla, analizarla y obtener nuevos beneficios tanto para el nadador como para la gestión por parte de la piscina.

Esta arquitectura, como cualquier otra orientada a la nube, ha de contener los siguientes elementos:



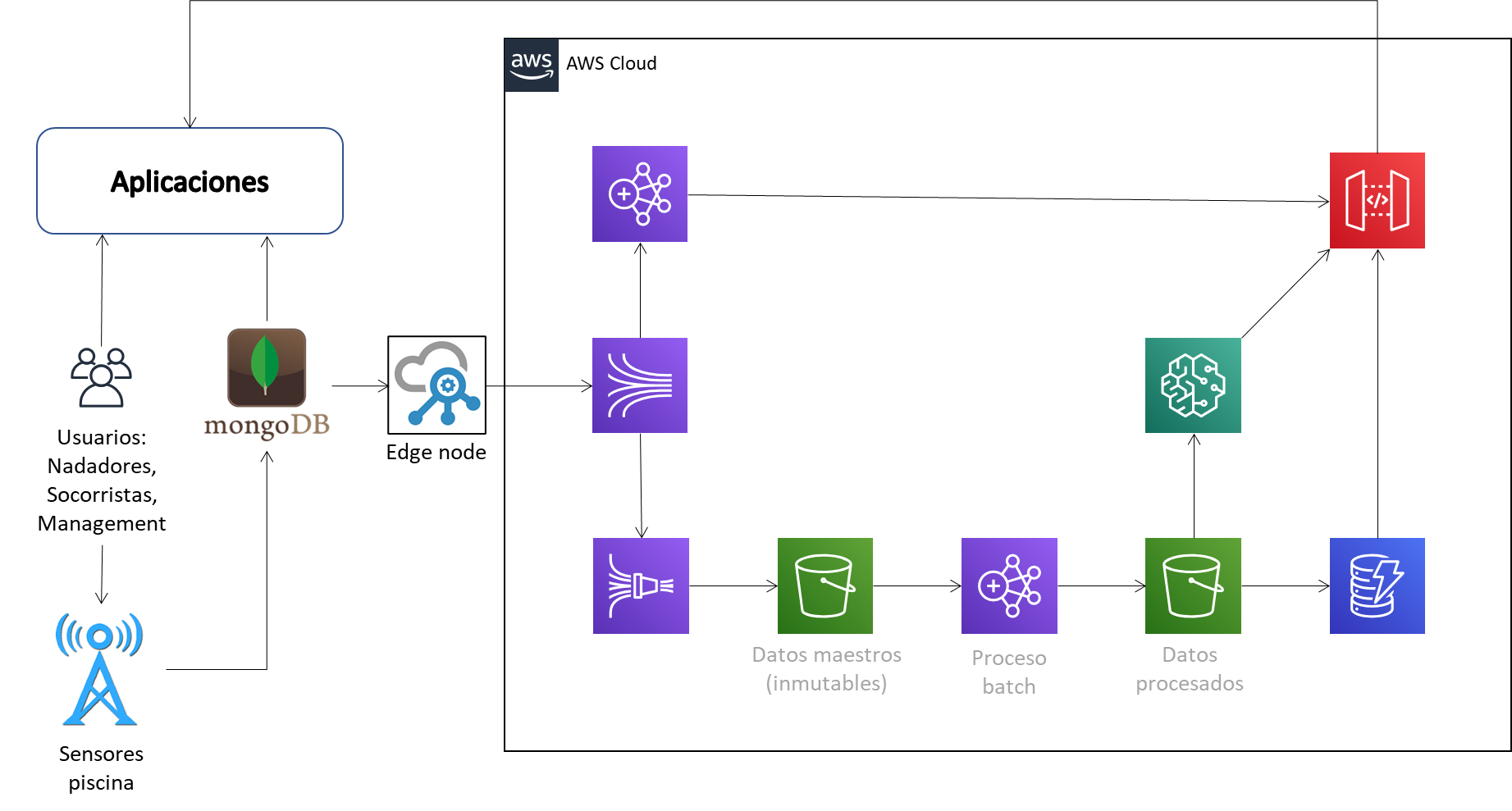
Para cubrir cada una de estas etapas las distintas compañías que operan en la nube proporcionan herramientas que podemos utilizar. A continuación, mostramos un gráfico donde destacamos las herramientas de las tres principales alternativas que hemos valorado para la realización de nuestro proyecto, Azure, AWS y Google Cloud:



Como podemos apreciar y por todos es conocido, las tres nubes disponen de las herramientas necesarias para llevar a cabo nuestro proyecto, cada una con sus fortalezas y sus debilidades. Se podría valorar, por tanto, un uso mixto para poder sacar partido de las principales fortalezas de cada una de ellas en determinados campos, pero hemos descartado de pleno esta opción.

La decisión de descartar el uso de varias nubes simultáneamente viene después de un análisis donde consideramos que implicaría incorporar no sólo complicaciones técnicas adicionales (necesidad de conocimientos amplios en cada nube), sino también de mantenimiento (implica un control más exhaustivo en cada una de ellas), seguridad (mantener y garantizar las políticas de seguridad de cada una internamente y para la comunicación entre ellas) y, como factor más determinante, precio (cualquier movimiento de información entre nubes tiene un coste alto que no vemos necesario abordar).

Una vez confirmado que sólo vamos a utilizar una de las nubes para el desarrollo de nuestro producto, teníamos que decidirnos por una de ellas. Puesto que la funcionalidad y precios de cada nube era muy similar, nuestro voto fue para **AWS** ya que es la nube en la que nos sentimos más cómodos desarrollando y programando. Por tanto, la arquitectura que pretendemos desarrollar a lo largo de este documento será la siguiente:

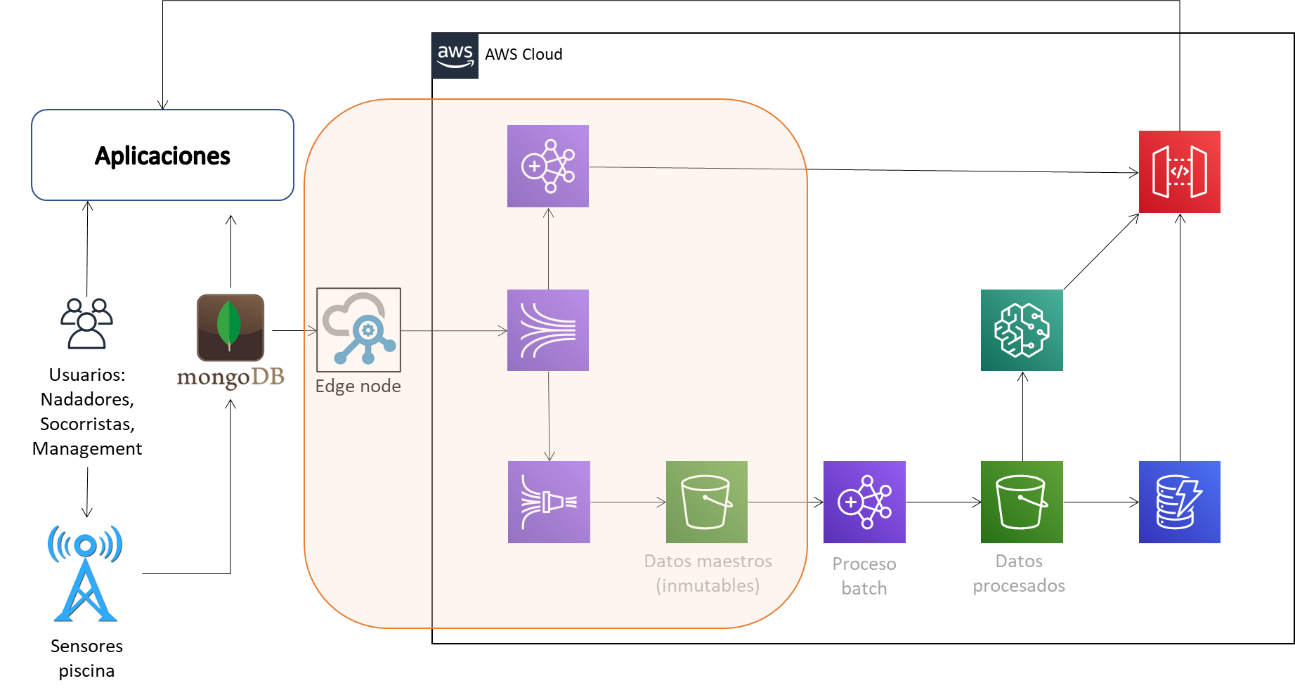


Antes de entrar en cada uno de las partes que conforman este esquema, hay que prestar atención a la necesidad de un **Edge node** encargado de gestionar la comunicación entre la piscina y la nube. Este elemento se tratará en más profundidad en el siguiente apartado: [3 Ingesta y almacenamiento del Dato](#_Ingesta_y_almacenamiento).

# Ingesta y almacenamiento del Dato

Este apartado tendrá como misión explicar en un nivel detallado el diseño y la explicación técnica de la arquitectura propuesta para las tareas de ingesta y almacenamiento del dato procedente de la piscina.

Tomando como referencia el esquema indicado en el apartado [**2.2 Arquitectura propuesta**](#_Arquitectura_propuesta), este apartado se centrará en los elementos remarcados a continuación:



## Razones para automatizar la ingesta de datos

Entendemos por ingesta a las maneras en las que se pueden obtener e importar datos, ya sea para uso inmediato o para ser almacenados. Importarlos también incluye el proceso de prepararlos para un análisis. En un sentido más amplio, la ingesta de datos puede ser entendida como un flujo dirigido entre dos o más sistemas que resulta en una operación fluida e independiente.

La ingesta puede ocurrir en tiempo real, tan pronto como la fuente los produce; o en lotes, cuando los datos son ingresados en cantidades específicas en periodos definidos.

Generalmente, tres pasos ocurren durante la ingestión de datos:

* **Extracción**: recolectar datos desde la fuente.
* **Transformación**: validar, limpiar y normalizar los datos asegurándose de su precisión y confiabilidad.
* **Carga**: colocar los datos en el silo o base de datos correcta para su análisis posterior.

Mientras los datos crecen, estos pasos se hacen más grandes y toman más tiempo. Las compañías pueden tener diversas fuentes de datos que funcionan las 24 horas del día. Automatizar estas tareas repercutirá beneficiosamente en varios aspectos como mejorar los objetivos del time-to-market, aumentar la escalabilidad y enfocar la atención en el trabajo necesario

Por tanto, y como resumen, la ingesta de datos es una tarea imprescindible en el funcionamiento hoy en día del Big Data y, como tal, hemos de configurarla y prepararla convenientemente para obtener el máximo rendimiento de ella.

## Edge node

El edge-node, o nodo borde, es la pieza de nuestro diseño que se encargará de gestionar la comunicación entre la parte on-premise del cliente y nuestra nube de AWS. Este componente se hace necesario para garantizar no sólo que la comunicación se realice en la forma en que la necesitamos, sino también para interferir lo menos posible en el funcionamiento actual del sistema, minimizando de esta forma los problemas técnicos derivados de cualquier implantación de este tipo.

Este componente debe formar parte de la red on-premise y tener igualmente conectividad con internet para poder enviar la información a la nube. En las funciones iniciales que le hemos establecido el nivel de procesamiento necesario es mínimo, por lo que este servicio podría realizarse incluso con un dispositivo tipo Raspberry PI, siempre que se encuentre conectada a la red WIFI. Debido a la simplicidad de la tarea, en el caso de requerirse una mayor capacidad de cómputo siempre estará disponible la opción de escalar de forma vertical.

El sistema garantizará que todos los mensajes que han de enviarse lo hagan, independientemente de que hubiera algún fallo en la comunicación o algún otro evento que impidiera el envío en primer lugar.

### Configuración

Este dispositivo funciona tanto con un entorno Windows como LINUX, ya que todos los componentes se han desarrollado en lenguajes compatibles con ambos sistemas.

Las credenciales de acceso a AWS se deberán configurar en este equipo, tal y como se indica en la guía de [**Opciones de los archivos de configuración y credenciales de AWS**](#_Bibliografía) que figura en la bibliografía al final de este documento**.** Consideramos que la forma más sencilla es mediante la creación del fichero **credentials** en la carpeta **.aws** del directorio de origen del usuario que realizará esta tarea. Este directorio de origen ,dependiendo de si es un sistema Windows o Linux, tendrá una ruta determinada, siendo %UserProfile% en Windows y $HOME o ~ (tilde) en sistemas basados en Unix.

El contenido del fichero, independientemente del sistema operativo, deberá tener una estructura similar a la siguiente:

*[default]*

*aws\_access\_key\_id=****ACCESS\_KEY***

*aws\_secret\_access\_key=****SECRET\_ACCESS\_KEY***

Siendo **ACCESS\_KEY** y **SECRET\_ACCESS\_KEY** los valores definidos en el IAM de AWS para gestión de acceso.

También se podrá configurar la región por defecto a la que se conectarán los componentes AWS, definiendo el valor en el fichero config de la misma carpeta. El contenido deberá tener una estructura similar a la siguiente:

[default]

region=***REGION\_NAME***

Siendo **REGION\_NAME** el valor de la región donde estarán los componentes, por ejemplo, us-east-1.

El sistema deberá tener instalado Python al menos en versión 3.7, así como los componentes boto3.

### Componentes

Para gestionar la comunicación entre el centro y la nube hemos desarrollado una serie de componentes en Python cuyo código fuente puede consultarse en la sección [**9. Código fuente**](#_Código_Fuente). El listado de componentes es el siguiente.

**Config.ini**

Fichero de configuración para indicar determinados parámetros a utilizar durante la conectividad. Este fichero deberá estar localizado en el mismo path que los componentes Python. Deberá contener los siguientes apartados:

* **LOGGER.** Información sobre el tipo de log a utilizar.
  + ***level***. Indicará el nivel de log que queremos. Las opciones disponibles son DEBUG, INFO y ERROR, indicadas de mayor a menor nivel de detalle sobre las acciones realizadas.
* **PARAMETERS.** Parámetros relacionados con elementos propios de la comunicación con Kinesis.
  + ***location\_id***. Registrará el campo location\_id del mensaje a enviar a Kinesis, reflejando en este caso el indicador de la piscina que está realizando el envío. Puesto que esta información ayuda a AWS a determinar el shard donde mandar la información, se podrá utilizar el ID de cada uno de los mensajes en su lugar.
  + ***kinesis\_stream***. Nombre del stream de Kinesis donde se ha de enviar la información de los mensajes.
  + ***retries.*** Número de reintentos para los envíos de paquete de mensajes a kinesis. El mínimo serán 0 y el máximo 5, siendo este último el valor por defecto.
  + ***buffer.*** Tamaño del buffer de mensajes a enviar en cada tanda. El límite superior viene dado por Amazon en 500 mensajes por envío, mientras que el inferior se ha establecido en 10. El valor por defecto será 500.
  + ***region.*** Región a utilizar en las conexiones. Si no se especifica se usará la definida por defecto en el fichero config de la carpeta .aws.

**Connectivity**

Ésta será la clase encargada de realizar el envío de mensajes al stream de Kinesis. Para que la comunicación con la nube se pueda realizar correctamente deberán estar registradas las credenciales de AWS en el equipo, como se explicó en los apartados anteriores. Además, este componente hace uso del fichero *config.ini* anterior, el cual deberá estar en el mismo path. Las funciones que pone a nuestra disposición esta clase son las siguientes:

* ***sendOneRecordKinesis****.* Permitirá realizar el envío de un mensaje individual al stream de kinesis.
* **sendRecordsKinesis**. Permitirá realizar el envío de más de un mensaje a la vez. Como se mencionó anteriormente, los componentes del SDK de Kinesis tienen la limitación sobre el número de mensajes que se pueden enviar en cada comunicación. Este valor, además, vendrá definido en el fichero de configuración. El procedimiento se encargará de crear y enviar paquetes con el tamaño indicado hasta finalizar con el total de los mensajes pasados como parámetro. Puntualizar que el número de reintentos indicado en el fichero de configuración aplicará para cada paquete que se envíe, no para el total. De esta forma, si se han de enviar por ejemplo 10 paquetes de datos, cada uno de ellos podrá utilizar hasta 5 reintentos (o el valor definido en el fichero de configuración).
* **sendCSVKinesis**. Esta función recibirá como parámetro el path con el nombre del fichero CSV cuyo contenido habrá de enviarse a Kinesis. El procedimiento leerá todos los registros del fichero CSV, los encapsulará y hará una llamada a la función sendRecordsKinesis para enviarlos.

Todos los métodos anteriores devolverán un diccionario que contendrá la siguiente información:

* + *status.* Valor booleano que indicará si el proceso se realizó correctamente (True) o no (False).
  + *message.* En caso de error (status = False), contendrá una explicación del error obtenido.

Como se indicó en la descripción del fichero de configuración, habrá varios niveles de log dentro de la ejecución de los procedimientos de esta clase. Estos logs se crearán con nombres basados en la hora actual y dentro de la carpeta “logs” situada en el mismo path de la clase. Si esta carpeta no existiera, se creará en tiempo de ejecución.

Como medida de seguridad, para garantizar que todos los mensajes fueron enviados correctamente a Kinesis a pesar de algún corte en la comunicación u otro evento que pudiera haber ocurrido, esta clase conservará un registro de todos los mensajes no enviados para proceder a su reenvío una vez se recupere la conectividad. Este registro se hará en memoria y se procederá al reenvío cada vez que se llame a alguna de las funciones de reenvío. Además, se guardará un registro en disco que se recuperará a memoria e intentará enviar cada vez que se llame esta clase.

**Serverqueue**

Esta clase creará un servidor que estará escuchando en un socket determinado por llamadas POST para realizar el envío de mensajes a Kinesis. Este servidor se apoya en el uso de la clase Connectivity anterior para gestionar el envío de datos desde el servidor a Kinesis.

Al crear la clase se podrán proporcionar como parámetros el nombre del host así como el puerto donde estará escuchando el proceso. Los distintos puntos de llamada serán los siguientes:

* **host:port/sendRecord**. Permitirá el envío de un único registro.
* **host:port/sendRecords**. Permitirá el envío de más de un registro cada vez.
* **host:port/sendCSV**. Permitirá el envío del contenido de un CSV pasado como parámetro.

Para garantizar que el envío se está realizando correctamente, los registros se introducirán en una cola y serán procesados secuencialmente por un thread específico que creará una clase Connectivity para el envío.

### Integración on-premise

Para realizar la integración con el funcionamiento actual del centro, en base a los componentes explicados en el apartado anterior (envío directo con la clase Connectivity o utilizando llamadas POST al servidor ServerQueue) se propondrán distintas alternativas al centro sobre cómo afrontarla:

1. Desarrollar un trigger en MongoDB para que cada vez que se inserte un registro, éste sea enviado directamente a la nube.
2. Desarrollar un demonio en MongoDB que monitorice la base de datos y cada cierto tiempo realice un envío de información a la nube.
3. Desarrollar un demonio en MongoDB que monitorice la actividad de MongoDB y cada cierto tiempo genere un CSV con los registros pendientes de enviar y lo envíe.
4. Crear un demonio ajeno a MongoDB que monitorice esta BD y cada cierto tiempo realice envíos de la información, bien generando un CSV intermedio o directamente desde memoria.

## Simulador piscina

Al no disponer de comunicación online con las piscinas de nuestro partner en esta fase del proyecto, decidimos crear un elemento que nos permitiera enviar información a nuestro stream para simular el estado de una piscina.

Este componente se ha desarrollado en Python, permitiendo simular la existencia de una piscina y sus nadadores, realizando el envío de posición de cada uno de ellos apoyándose en el componente de conectividad que se explicó en el apartado anterior. Se creó adicionalmente un notebook de Jupyter para hacer uso de estos componentes y tener una representación visual de la situación.

A continuación, explicaremos el funcionamiento y cometido de cada uno de los componentes, pudiendo consultar su código en el apartado [**9. Código fuente**](#_Código_Fuente) al final de este documento.

**Nadador**

Esta clase simulará la actividad de un nadador, el cual tendrá una serie de características únicas que enumeramos a continuación:

* **tagID**. Proporcionado por la piscina, identificará unívocamente al dispositivo que utilizará el nadador para enviar información a las antenas.
* **tagFreq**. Frecuencia, indicada en milisegundos, en que se enviará información sobre la posición del nadador.
* **Tiempo**. Tiempo, indicado en segundos, durante el que el nadador estará en la piscina.
* **Estilo.** Valor aleatorio sobre el estilo de nado, el cual representará la velocidad del nadador. Los posibles valores son crawl (60%), braza (30%) y mariposa (10%).
* **Velocidad.** Valor determinado por el estilo de natación definido. Los valores base serán de 1 m/s para crawl, 0’8 m/s para braza y 0’9 m/s para mariposa (la mitad del valor registrado como récord mundial), siendo estos valores incrementados/decrementados aleatoriamente para tener distintas velocidades dentro del mismo estilo de nado.
* **Posición.** Cada nadador tendrá una posición inicial dentro de la piscina y una dirección de nado, dependiendo de por dónde “entre” a la piscina. Este valor vendrá dado por la piscina.
* **Modo de registro.** La generación de mensajes se podrá realizar para un envío online (generando JSON) o sólo para generar ficheros de log (string plano).

La clase creará un thread que cada vez que tenga que generarse un registro (indicado por la variable de frecuencia) introduzca éste en una cola. Esta cola será gestionada por la piscina para realizar la tarea concreta de registro de log o de envío online. Cuando se cumpla el tiempo en que el nadador estará en la piscina, se marcará el nadador como inactivo.

**Piscina**

Esta clase simulará la actividad de una piscina. Se deberán indicar los siguientes parámetros al crear la clase:

* **Largo y ancho de la piscina**. Medidas de la piscina a representar.
* **Ancho de la calle**. Permitirá calcular el número de nadadores por calle que se podrán simular. El valor por defecto es de 2’5 m.
* **Refresco**. Valor en milisegundos sobre la frecuencia con que cada nadador de la piscina generará un registro de posición. El valor por defecto serán 40 ms.
* **Modo de registro**. Podrá ser en modo log (CSV) o modo online. El valor por defecto será CSV.
* **Tiempo vacía**. Indicará el tiempo que la piscina podrá permanecer vacía (sin nadadores) antes de considerar que nadie más va a nadar y la piscina “cierre”, dando el proceso por finalizado.

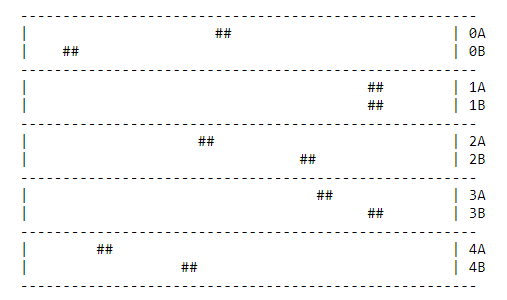
En función de los parámetros facilitados se determinará el número simultáneo de nadadores que podrá haber en la piscina y creará los TagID’s que repartirá entre los nadadores que entren.

La clase permitirá la entrada de nadadores a su piscina hasta el máximo permitido, monitorizando su actividad y gestionará la cola de mensajes para realizar la tarea indicada en el modo de registro (CSV o envío online). También comprobará el estado de los nadadores para que, si uno de ellos termina su actividad, vaciar su calle y permitir que otro nuevo nadador pueda ocupar su lugar.

**Gestión de piscina**

Se creó un notebook de Jupyter que permitirá, haciendo uso de las clases anteriores, gestionar una piscina y tener una visión gráfica del funcionamiento. El notebook está preparado para recibir las dimensiones de la piscina, el número de nadadores y el rango de tiempos de natación, así como el modo de registro.

El Notebook crea una instancia de la piscina e incluye tantos nadadores como se indiquen. Mostrará una representación gráfica del estado de la piscina, indicando la posición de cada uno de los nadadores en cada calle.



## Amazon S3

Amazon Simple Storage Service (Amazon S3) es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento líderes en el sector, diseñado para ofrecer una durabilidad del 99,999999999%. Por todos estos motivos junto a la compatibilidad e integración con las diferentes herramientas de la nube AWS decidimos utilizarlo en nuestro proyecto.

En nuestro proyecto, y como se verá en este documento, haremos varias veces referencia a este servicio ya que es donde se apoyarán otras de las herramientas utilizadas.

En nuestra arquitectura del proyecto, y dentro de la rama de ingesta, recordemos nuestro primer punto dedicado al almacenamiento de información:

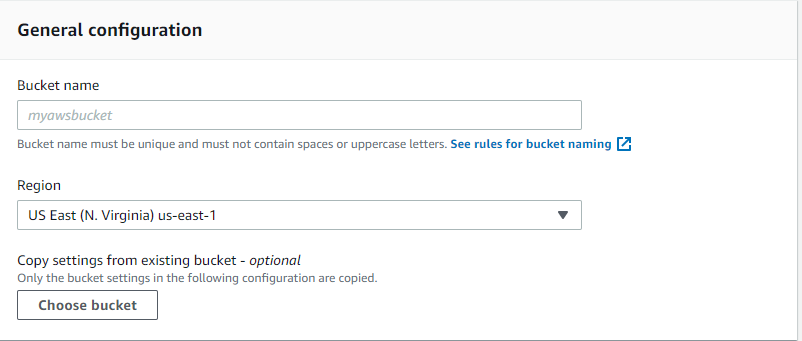


Como podemos ver, para nuestro proyecto deberemos tener un punto donde poder almacenar toda la información recibida desde el cliente y que ésta permanezca inalterada tal cual fue recibida. De esta forma no condicionaremos los posibles procesamientos que se hagan de esta información.

### Creación del bucket

Para proceder a crear este repositorio de información, o **bucket**, accederemos a la parte de AWS específica y pulsaremos sobre el botón  para establecer los criterios de nuestro repositorio.

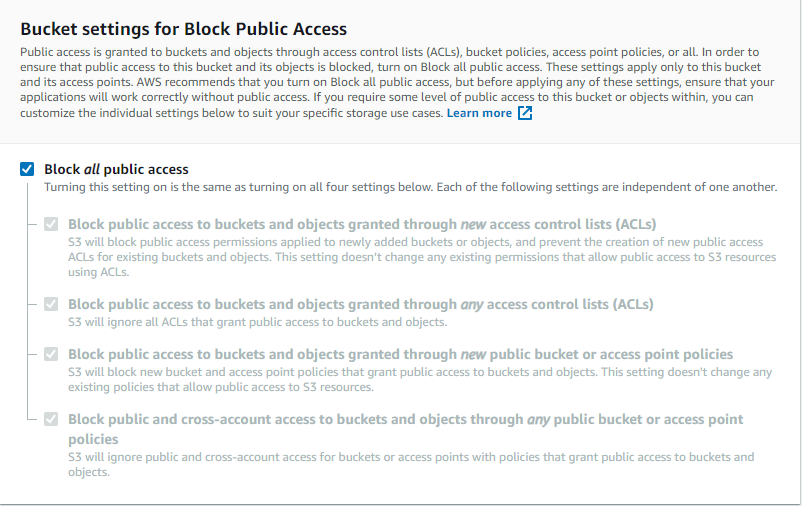
**Configuración general**



Deberemos establecer un nombre para el bucket donde registraremos la información. Este nombre debe ser único en todo el universo de AWS y no podrá contener ni espacios ni mayúsculas.

Habrá que definir, además, la región donde este bucket será creado. Este factor resulta importante ya que a la hora de procesar información el coste de acceder a esta información no será el mismo si accedemos desde la misma región o desde otra.

**Configuración de acceso público**



Por defecto la opción es la de no disponer de esta información al exterior y es la que mantendremos en nuestra configuración.

**Otra configuración**

Existen otras opciones de configuración que mantendremos en los valores por defecto, como son los relacionados al versionado de elementos (mantener distintas copias de un mismo fichero en distintas partes del bucket), tags o encriptación de la información. Este último punto quedaría a decisión del cliente, pero en nuestra opinión no sería necesaria ya que no hay ningún tipo de información sensible que pudiera comprometer la identidad de los datos.

### Estructura de carpetas

Las carpetas y directorios no existen realmente dentro de los buckets S3. La estructura de ficheros es sólo un nivel plano de un único nivel de ficheros. La ilusión de estar trabajando con directorios viene en base a la nomenclatura de los ficheros del tipo dirA/dirB/fichero. La interfaz de S3, no obstante, nos muestra la información como si nos estuviéramos moviendo por un conjunto de carpetas para facilitar la comprensión de la información.

En nuestro caso, crearemos una única carpeta con la información recibida desde la piscina. Llamaremos a esta carpeta **incoming** y más adelante explicaremos de qué manera vamos a almacenar la información aquí.

### Coste

El servicio de almacenamiento que ofrece S3 tiene unos costes asociados que, si bien no son elevados, sí que hay que tener en cuenta para el presupuesto. Pueden ser consultados en la página de [**Precios de Amazon S3**](#_Bibliografía). Resumiendo, los costes de los servicios S3 vendrían dados por:

* Almacén de la información.
* Solicitudes de escritura/lectura.
* Transferencia de datos entre regiones o fuera de la nube.
* Servicios de administración y replicación

Según los datasets que hemos recibido de ejemplo de nuestro partner, cada día se generan en torno a unos 700 MB de información. Haciendo números, vemos que cada mes almacenaríamos unos 21GB de información, lo que supondría unos 50 céntimos al mes de coste de almacenamiento, 6€ al año. El resto de gastos harían que, como mucho, incrementáramos el gasto a unos 10€ en total al año.

La información que almacenaremos en este bucket, una vez procesada es muy probable que no necesitemos accederla más que para futuras auditorías o controles. Por tanto, pasado cierto tiempo la información se podría mover a un tipo de almacenamiento Glacier, con un coste menor pero eso sí, con un tiempo de acceso mayor.

## Kinesis data stream

Para realizar la gestión de la ingesta, sabíamos que íbamos a necesitar una plataforma distribuida de transmisión de datos que permitiera publicar, almacenar y procesar flujos de registros, y suscribirse a ellos, en tiempo real. Esta plataforma debería ser capaz de manejar flujos de datos de varias fuentes y distribuirlos a los diversos usuarios. En resumen, necesitábamos una herramienta que fuera capaz de transferir cantidades enormes de datos, no sólo entre dos puntos sino también a cualquier otro lugar que lo requiriera, todo al mismo tiempo.

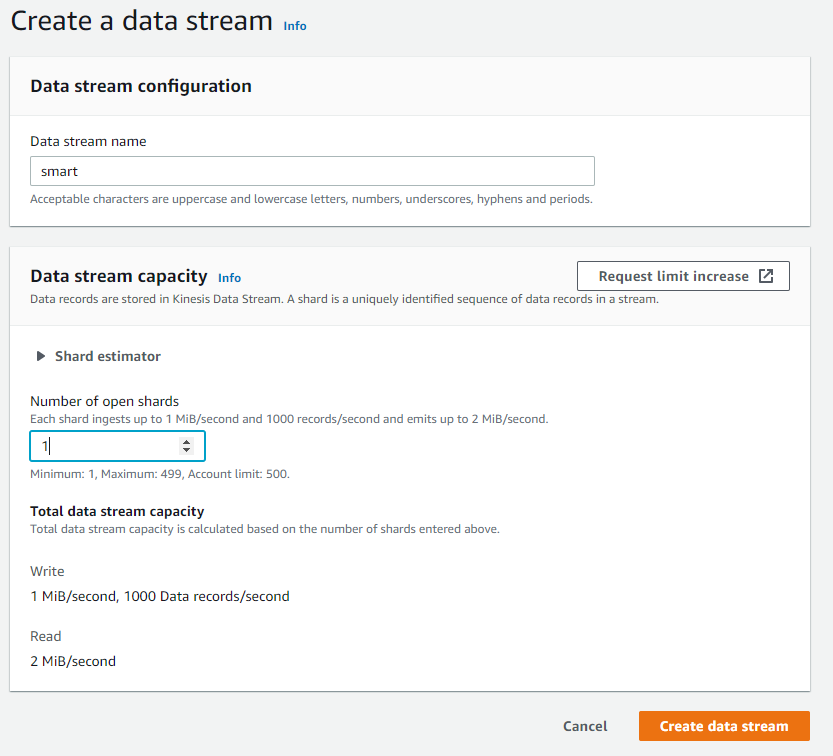
En esta primera fase del proyecto la información no la recibiremos en la nube directamente desde los dispositivos IoT instalados en la piscina, sino que serán enviados desde el Edge node, proceso explicado en el punto anterior. Sin embargo, pensando en las posibilidades de mejora y en un futuro envío de más información desde distintos puntos, necesitábamos disponer de una herramienta con capacidad para recibir información desde más de una fuente de datos.

Cuando pensamos en este tipo de arquitectura irremediablemente pensamos en Apache Kafka. Aunque esta aplicación se puede desplegar y utilizar en AWS, finalmente nos decidimos a utilizar una de las aplicaciones nativas de esta nube como es Kinesis, ya que su integración con las distintas herramientas de esta nube es total. Además, su configuración resulta más sencilla al estar más automatizada y su coste es también inferior al de la implementación de Kafka. Por todos estos motivos, la parte de ingesta de nuestro proyecto se realizará con Kinesis.

A lo largo de este apartado explicaremos los pasos seguidos para configurar este producto, dando explicación a los motivos que nos llevan a tomar cada decisión.

### Creación y configuración

Para crear es Data Stream, accedemos al servicio en AWS y completamos la información necesaria para su creación:



El nombre del Data Stream se compondrá únicamente por letras (mayúsculas o minúsculas), números, guiones altos o bajos y puntos. El nombre, además, debe ser único dentro de la región.

El número de shards determinará la capacidad de lectura y escritura del stream siendo el mínimo 1, lo que equivaldría a 1MB/segundo (1000 registros por segundo) y 2MB / segundo de lectura.

En base a la información proporcionada por el centro, cada piscina genera en torno a unos 2 millones de registros al día. Suponiendo que la piscina estuviera abierta únicamente 8 horas, eso implicaría que cada segundo se generarían unos 70 registros por segundo. Cada registro tiene un peso de unos 250 bytes, lo que haría una carga total de 70 registros / segundo x 250 bytes = 18 KB / s, muy lejos del límite de 1MB que nos proporciona un shard.

Por tanto, únicamente utilizaremos un shard para nuestro proyecto. Si más adelante se incorporaran nuevas piscinas o más dispositivos se tendría que valorar incrementar este valor.

Otros valores que se podrían configurar irían relacionados con la encriptación de la información (como en el caso de S3) y, especialmente interesante, la retención de información en el stream. Este parámetro determinará durante cuánto tiempo la información del stream persistirá en éste. El valor mínimo, que será el que utilicemos para este proyecto, es de 24 horas. El máximo se podrá establecer en 168 horas (una semana).

### Coste

Amazon Kinesis Data Streams aplica un modelo sencillo de precios por uso. No requiere costos iniciales ni cuotas mínimas y solo es necesario pagar por los recursos que se consuman. Los precios se basan en dos variables principales: partición de hora y unidad de carga PUT, y variables opcionales para la distribución ramificada mejorada y la retención de datos ampliada. El coste se puede consultar en la página de [**Precios de Amazon Kinesis Data Streams**](#_Bibliografía).

En nuestro caso, al únicamente tener un shard activo, el coste sería aproximadamente de unos 10€ al mes (24 horas x 30 días x 0,015 USD / hora). La parte correspondiente a las unidades de carga sería casi despreciable (unos 50 cts al mes).

## Kinesis Data Firehose

Amazon Kinesis Data Firehose ofrece la manera más sencilla para cargar datos de streaming de manera fiable en datalakes, almacenes de datos y servicios de análisis. Se trata de un servicio completamente administrado cuya escala se ajusta de manera automática para adaptarse al nivel de procesamiento de sus datos. Además, no requiere administración continua. También puede procesar por lotes, comprimir, transformar y cifrar las transmisiones de datos antes de cargarlas a fin de minimizar el volumen de almacenamiento utilizado y aumentar el nivel de seguridad.

Debido a las características que proporciona este servicio nos pareció la mejor opción para volcar los datos desde el stream de entrada hacia el bucket en S3 que creamos en el punto 3.3.

### Configuración

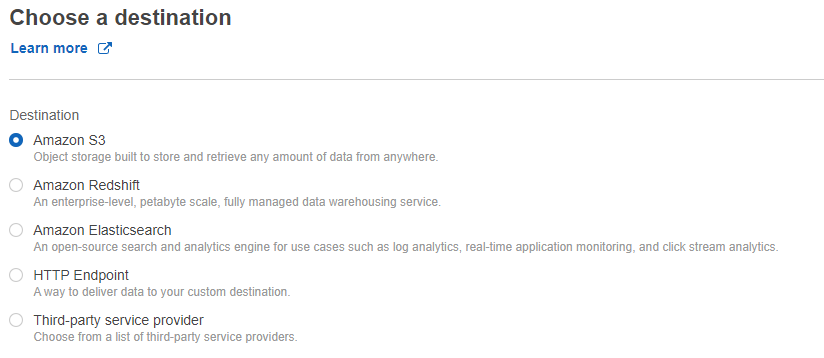
Para crear un nuevo elemento hemos de definir el nombre, el cual deberá ser único dentro de la cuenta desde la que estemos trabajando.

Una vez hecho esto, hay que elegir el origen de la información. Esto puede ser flujo directo de información (envío directo, IoT, Cloudwatch logs, etc) o un stream de Kinesis. Ésta será la opción que elegiremos, tomando nuestro stream de datos creado en el punto anterior como origen de la información.

En el siguiente punto deberemos definir si queremos aplicar algún tipo de transformación de los registros origen mediante Lambdas. Esto no lo vamos a aplicar puesto que realizaremos la limpieza del dato mediante otros procesos más adelante. Más información en el apartado [**4. Limpieza y preparación del dato**](#_Limpieza_y_Preparación).

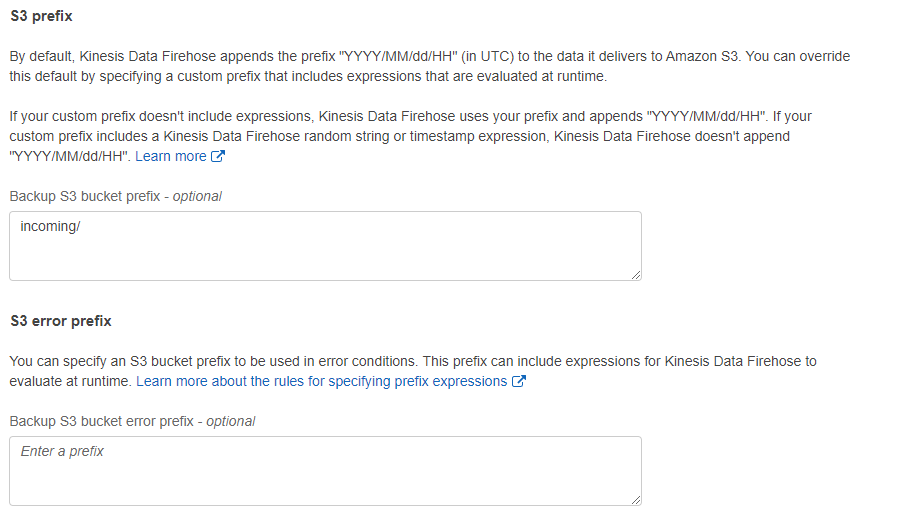
También tendremos la opción en esta pantalla de definir si queremos algún tipo de transformación de formato automático a un formato más eficiente como es parquet. Tampoco en este caso haremos uso de esta opción por el mismo motivo que antes.

A la hora de determinar el destino de la información tendremos varias opciones a nuestra disposición:



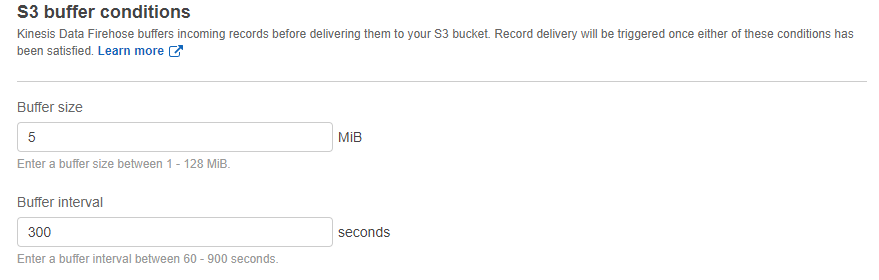
Según nuestra arquitectura, toda la información que recibamos de la piscina irá almacenada en nuestro bucket de S3 desde donde más adelante será procesada. Por tanto, elegimos esa opción y escogemos el bucket S3 que creamos anteriormente.

Además del bucket, podemos definir un prefijo para localizar la información. Es en este caso donde definiremos el “directorio” que creamos cuando definimos nuestro bucket S3, de forma que toda la información recibida se registre bajo ese formato. La definición sería de la siguiente manera:



Por defecto y, salvo que lo configuremos de una forma diferente el sistema, además de poner ese “prefijo” a nuestros mensajes recibidos el servicio firehose creará una estructura de carpetas con la estructura **incoming/YYYY/MM/DD/HH**, siendo YYYYMMDDHH los valores del timestamp de llegada de los mensajes. Esto es importante considerarlo ya que no se trata del timestamp que lleva dentro cada mensaje sino de uno estimado por Kinesis. Por tanto, si vamos a hacer uso de esta información con algún crawler de Glue o explotar esta información directamente desde la estructura de carpetas, que lo tengamos en consideración.

Por último, habrá que definir las condiciones del buffer de S3:



El sistema Firehose actuará cuando se cumpla alguna de esas dos condiciones, es decir, que la cantidad de información llegue a un determinado volumen (5 MB por defecto) o haya pasado un tiempo específico (300 segundos en el ejemplo). Con el fin de tener la información disponible en nuestro S3 lo antes posible, configuraremos los valores a 1MB y 60 segundos.

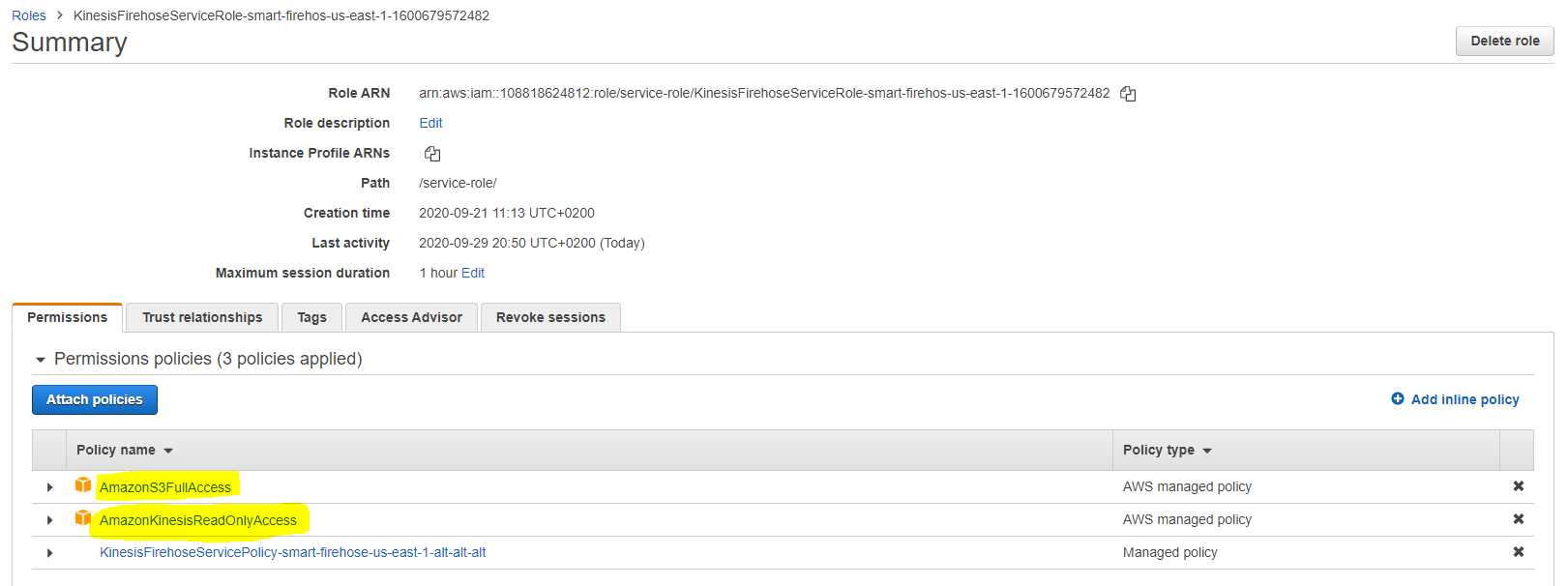
Opcionalmente podremos asignar compresión y encriptación de los datos, en el caso de que queramos tener la información más protegida y/o comprimida. Teniendo en cuenta el coste del espacio y el uso que le vamos a dar, no lo consideramos necesario.

### Configuración de accesos

Una vez creado el Firehose debemos proporcionarle permisos de acceso tanto al Data Stream (lectura) como al bucket S3 para poder volcar los datos. Este proceso lo haremos accediendo al rol del elemento en IAM y asignándole las siguientes políticas:

* AmazonS3FullAccess
* AmazonKinesisReadOnlyAccess

La foto final debería ser algo así:



### Coste

Se pueden consultar todas las tarifas en la página de [**Precios de Amazon Kinesis Data Firehose**](#_Bibliografía)**.**

El coste de este servicio es muy bajo teniendo en cuenta el nivel de datos que vamos a manejar. Teniendo en cuenta los 21 GB mensuales calculados antes el coste serían unos 50 céntimos al mes. Si decidiéramos hacer algún tipo de transformación el coste podría incrementarse, pero no es nuestro caso.

# Limpieza y Preparación del Dato

Texto

# Analítica y Modelo de Aprendizaje

Texto

# Servicio de información y visualización

Texto

# Propuestas de mejora

# Resumen Final y Conclusiones

Texto

# Código Fuente

## connectivity.py

## serverqueue.py

## piscina.py

## nadador.py

## SMART\_Gestión\_Piscina.ipynb

# Bibliografía

* **Infografía Big Data: las 7 V**. <https://www.iic.uam.es/innovacion/big-data-infografia-7-v/>
* **4 razones para automatizar la ingesta de datos**. <http://www.icorp.com.mx/blog/automatizar-ingesta-de-datos/>
* **Opciones de los archivos de configuración y credenciales.** <https://docs.aws.amazon.com/es_es/cli/latest/userguide/cli-configure-files.html>
* **Precios de Amazon S3.** <https://aws.amazon.com/es/s3/pricing/>
* **Precios de Amazon Kinesis Data Streams.** <https://aws.amazon.com/es/kinesis/data-streams/pricing/?nc=sn&loc=2>
* **Precios de Amazon Kinesis Data Firehose.** <https://aws.amazon.com/es/kinesis/data-firehose/pricing/>