UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

# Arquitecturas y Modelos Económicos de Aplicaciones Serverless

Trabajo Fin de Máster

**Máster Universitario en Gestión de la Información**

**Autora**: Luiza Petrosyan

**Tutor**: Germán Moltó Martínez

Curso 2019/2020

# Resumen

Serverless es uno de los últimos avances en Cloud Computing. El concepto ha surgido como un nuevo paradigma para el despliegue de aplicaciones y servicios sin necesidad de una gestión explícita de servidores. Representa una evolución de los modelos de servicio en la nube, y es una evidencia de cómo han madurado y se han ampliado las tecnologías en la nube y de su enorme potencial.

En este trabajo final de máster vamos a examinar el fenómeno de la computación Serverless y las plataformas de este tipo existentes en la industria, sus arquitecturas y modelos económicos. Identificaremos sus características, y presentaremos un caso de uso de una aplicación web que permite una visualización de datos medioambientales de la ciudad de Madrid, utilizando los servicios Serverless de Amazon Web Servicies.

**Palabras clave:** Serverless, arquitectura, Cloud Computing, servicios, proveedor, aplicaciones.

# Agradecimientos

Me gustaría aprovechar estas primeras palabras para agradecer a todas aquellas personas que han hecho posible este trabajo.

En primer lugar, a mi tutor, Germán Moltó Martínez por guiarme durante el desarrollo. También quiero agradecer a Alberto García Giménez por su aportación de conocimientos en la parte práctica del proyecto.

# Índice

# Índice de Figuras

# Índice de Tablas

# 1. Introducción

## 1.1. Motivación

Serverless es un modelo poco conocido, incipiente que aporta grandes cambios en informática, como aportaron en su momento Virtualización y Cloud Computing.

Hace años la tecnología Cloud supuso una revolución al eliminar la dependencia de la gestión física de los servidores, actualmente el Serverless supone una nueva revolución al eliminar la necesidad de la gestión lógica de esos servidores, haciendo posible por primera vez la implementación de soluciones informáticas completas sin necesidad de gestión física ni lógica de máquinas.

En el máster de gestión de la información hemos estudiado e investigado las diferentes tecnologías que nos permiten llevar al cabo tareas análisis de datos desde la pequeña escala hasta grandes volúmenes como Big Data.

Dentro de las tecnologías investigadas irrumpe la arquitectura Serverless como nuevo paradigma de computación en la nube con grandes ventajas respeto a tecnologías precedentes.

La motivación para realizar este TFM ha sido analizar y comparar el paradigma Serverless respecto a arquitecturas consolidadas como es la infraestructura como servicio, que actualmente es el estándar de la industria.

Una motivación constante de las compañías es la eficiencia en el uso de sistemas de computación y la reducción de costes derivados de los mismos, especialmente los de recursos humanos. Gracias al alto nivel de abstracción de Serverless, observaremos como esta nueva tecnología puede simplificar y reducir enormemente los costes de implementación y producción en determinadas aplicaciones, y por lo tanto facilitar el acceso a funcionalidades complejas que antes solo estaban al alcance de grandes corporaciones.

Todos estos factores pueden convertir a esta tecnología en un estándar de la industria y en una arquitectura de implantación masiva.

## 1.2. Objetivos

El objetivo de este trabajo final de máster es analizar el paradigma de Serverless Computing y sus arquitecturas existentes, comparando con una arquitectura tradicional de computación en la nube y sus modelos económicos. Después nos centraremos en la implementación de una aplicación web dedicada para una visualización de datos en tiempo real, creando un modelo de arquitectura Serverless y veremos los costes incurridos durante el desarrollo del proyecto para evaluar su rentabilidad.

## 1.3. Organización

La organización de este documento está dividida en 4 capítulos que aportarán siguientes conocimientos.

Capítulo 1: Introducción.

Capítulo 2: Recopilación de información relacionadas al Cloud Computing y Serverless Computing, los servicios existentes y los proveedores líderes en el mercado, con sus modelos económicos.

Capítulo 3: Analizaremos las arquitecturas Serverless existentes y sus modelos de coste comparado con las arquitecturas no-Serverless. Termina el capítulo con un caso práctico desarrollado.

Capítulo 4: Conclusiones.

# 2. Tecnologías Relacionadas

## 2.1. Cloud Computing

Cloud Computing es un paradigma en evolución. Irrumpe en la industria tecnológica a partir de finales de 2007 debido a su capacidad para ofrecer infraestructuras de TI flexibles y dinámicas, entornos de computación garantizados con calidad de servicio y servicios de software configurables [1]. Como se ve en la Figura 1, Cloud Computing (línea azul) que es habilitado por la Tecnología de Virtualización (línea roja) ha superado a Grid Computing, que es una infraestructura distribuida de recursos de computación y almacenamiento usada principalmente para el ámbito científico (línea amarilla). Actualmente se han propuesto numerosos proyectos de la industria y la academia, con la iniciativa de investigación conjunta para diferentes soluciones en Cloud Computing sobre todo con la aparición de nueva capa de servicio FaaS (Function as a Service), de la cual vamos a hablar más profundamente en los siguientes párrafos de este trabajo.

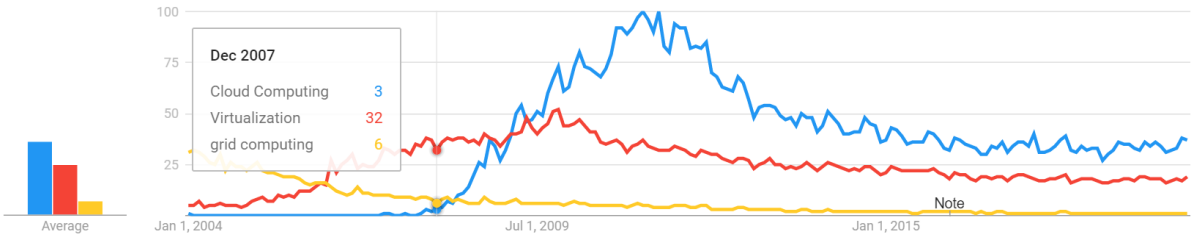


*Figura 1. Cloud Computing en Google Trends desde 2004 hasta presente.*

El Instituto Nacional de Estándares y Tecnología de E.E.U.U. define Cloud Computing de la siguiente manera [1]: ***"Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."***

Según NIST [1], Cloud Computing se puede caracterizar como:

- *Autoservicio en demanda*: El consumidor tiene la capacidad de aprovisionar cómputo de manera autónoma, pagando por el tiempo de uso de los servidores y el almacenamiento en la red de forma automática sin necesidad de interacción humana con cada proveedor de servicios [1].

- *Amplio acceso a la red:* Las capacidades están disponibles a través de la red y se accede a ellas a través de tecnología estándar que promueven el uso como, móviles, tabletas, portátiles, estaciones del trabajo y servidores [1].

- *Agrupación de recursos:* Los recursos informáticos del proveedor se combinan para servir a múltiples consumidores utilizando un modelo de múltiples inquilinos, con diferentes recursos físicos y virtuales asignados dinámicamente y reasignados de acuerdo con la demanda del consumidor. El cliente no tiene conocimiento sobre la ubicación exacta de los recursos proporcionados, solo se especifica la ubicación hasta el nivel de país, estado o centro de datos. Los ejemplos de recursos incluyen almacenamiento, procesamiento, memoria y ancho de banda de red [1].

- *Elasticidad rápida:* Las capacidades pueden ser aprovisionadas y liberadas elásticamente. En algunos casos automáticamente, para escalar rápidamente hacia fuera y hacia dentro de acuerdo con la demanda. Para el consumidor las capacidades para el aprovisionamiento a menudo pueden ser ilimitadas y pueden asignarse en cualquier cantidad en cualquier momento [1].

- *Servicio medido:* Los sistemas en la nube controlan y optimizan automáticamente el uso de los recursos al aprovechar una capacidad de medición en algún nivel de abstracción apropiado para el tipo de servicio. El uso de recursos puede ser monitorizado controlado e informado tanto por el proveedor como para el consumidor de servicio utilizado [1].

El modelo de nube definido por NIST [1] se compone de cuatro modelos de Implementación (Figura 2):

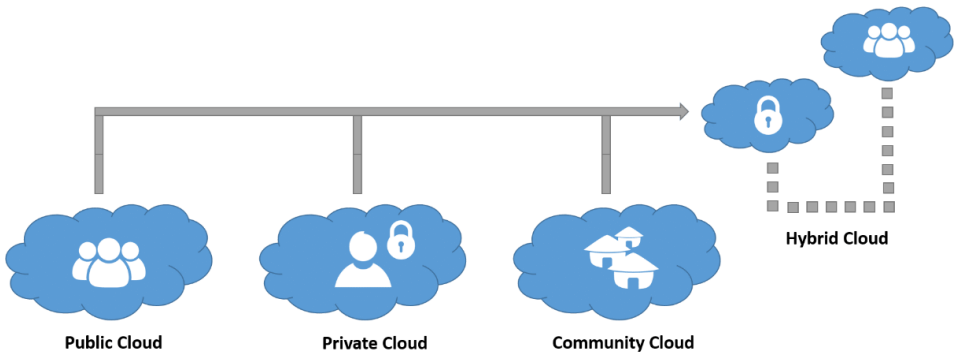


*Figura 2. Diagrama de estructuras de despliegue en la nube. Fuente: uniprint.net*

- **Cloud Privado**
  La infraestructura de la nube está prevista para el uso exclusivo por una sola organización que incluye múltiples consumidores (p.ej. unidades de negocios). Puede ser la propiedad, administrado y operado por la organización, un tercero

o alguna combinación de ellos y puede existir on u off premises (en los servidores locales y en la nube) [1].

La infraestructura de la empresa típicamente se encuentra detrás de un cortafuegos al que se puede acceder a través de conexiones cifradas. El pago a menudo se basa en un modelo de tarifa por unidad de tiempo.
Las nubes privadas tienen la ventaja importante de poder proporcionar niveles de seguridad y privacidad porque la infraestructura está dedicada a un solo cliente.

Una de las desventajas de las nubes privadas es la responsabilidad de administración de sus propias plataformas de desarrollo y aplicaciones de software en la infraestructura de los proveedores de servicio en la nube. Mientras esto le da al negocio un control sustancial de parte del software, tiene el costo de tener que entrenar personal de TI que pueda manejar la implementación en la nube de la empresa.
También tienen desventajas adicionales de que suelen ser más caras, en el caso de optar una aproximación off premises la empresa se limita a usar la infraestructura especificada en su contrato con el proveedor.

- **Cloud Comunitario**
  La infraestructura de la nube se proporciona para el uso exclusivo de una comunidad especifica de consumidores de organizaciones que tienen inquietudes compartidas (p.ej. misión, requisitos de seguridad, políticas, etc.). Puede ser de propiedad, administrada y operada por una o más de las organizaciones de la comunidad, un tercero o alguna combinación de ellas y puede existir on u off premises [1].

  Las nubes comunitarias son una opción atractiva para las empresas en los sectores sanitario, financiero o legal que están sujetos a un estricto cumplimiento normativo. También son adecuados para administrar proyectos comunes que se benefician de compartir aplicaciones de software o plataformas de desarrollo de la comunidad.

- **Cloud Público**
  La infraestructura de la nube está prevista para uso abierto por el público en general. Puede ser la propiedad, administrado u operado por una organización empresarial, académica o gubernamental, o una combinación de ellos. Normalmente las nubes públicas se basan en instalaciones de hardware masivas distribuidas por el país o por todo el mundo [1].

  Las nubes públicas son excelentes para las organizaciones que requieren administración de la infraestructura donde se ejecutan las aplicaciones y las diferentes aplicaciones que usan los usuarios.

Las nubes públicas son bastante rentables ya que la empresa sólo paga por los recursos informáticos que utiliza. Además, la empresa tiene acceso a una infraestructura moderna sin tener que comprarla y contratar personal de TI para la instalación y el mantenimiento.

Algunos proveedores de las nubes públicas pueden tener desventajas, ya que las provisiones avanzadas de seguridad y privacidad puede estar más allá de sus capacidades. Por ejemplo, relacionado con requisitos de cumplimiento normativo ya que sus inquilinos puede que compartan la misma infraestructura informática [2]**.** Sin embargo, hay ejemplos demostrando lo contrario, ofreciendo soluciones para ejecutar las máquinas virtuales sobre una infraestructura aislada (cloud privado).
Aunque las nubes públicas son adecuadas para alojar plataformas de desarrollo o aplicaciones web y para el procesamiento de Big Data que impone una gran demanda de recursos informáticos.

- **Cloud Híbrido**
  La infraestructura de la nube es una combinación de dos o más infraestructuras distintas de la nube (privado, comunidad o público) que siguen siendo entidades únicas, pero están unidas por una tectología patentada o estandarizada que permite la portabilidad de datos y aplicaciones (p.ej. cloud bursting, una configuración que se establece entre una nube privada y una nube pública para enfrentarse a los picos de carga) [1].

  Una nube híbrida es excelente para la escalabilidad, flexibilidad y la seguridad. Uno de los ejemplos sería una nube pública para interactuar con los clientes, mientras mantiene sus datos protegidos a través de una nube privada.

  Las nubes híbridas son adecuadas para llevar a cabo operaciones de Big Data con datos no confidenciales en la nube pública, mientras mantienen los datos confidenciales en la nube privada. Ofrecen también a las empresas la ejecución de sus aplicaciones públicas o plataformas de desarrollo intensivas en la parte pública mientras sus datos confidenciales permanecen protegidos en la nube privada [1].

### 2.1.1. Servicios Principales

Recopilar una comparación de servicios en la nube es una tarea de enormes proporciones en un entorno que evoluciona muy rápido. Hay miles de servicios en la nube, cientos de proveedores de servicios en la nube y docenas de proveedores de Infraestructura como Servicio (IaaS) que ofrecen modelos de precios de pago por uso, cada uno de los cuales cambia con frecuencia y actualiza sus porfolios.

Los mismos pertenecen a unos modelos de servicios que hoy en día se han convertido como parte integrante de la terminología de Cloud Computing.

*Modelos de servicio:*

Los servicios de Cloud Computing se dividen en 4 categorías:

- **Infrastructure as a service (IaaS)**
- **Platform as a service (PaaS)**
- **Functions as a Service (FaaS)**
- **Software as a service (SaaS)**

Estas a veces se denominan pilares de Cloud Computing (Figura 3).

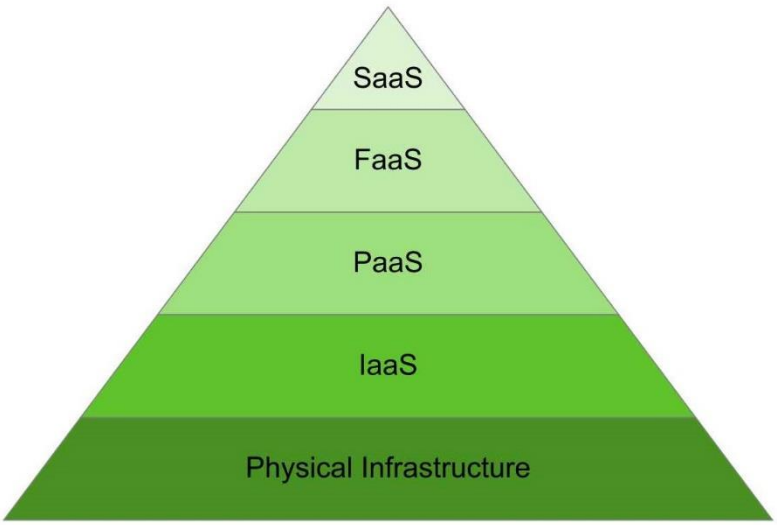


*Figura 3. Los modelos de servicio. Fuente: Google Images*

**Infrastructure as a Service (Iaas)** - es la categoría más básica de servicios de Cloud Computing, que permite alquilar infraestructura TI de un proveedor de la nube con el sistema de pago por uso.

Como un ejemplo de IaaS el servicio de cómputo. Se refiere a los servicios que ofrecen los proveedores como despliegue y la automatización de las instancias, las máquinas virtuales, contenedores, ajuste de escala, manejo de aplicaciones, funciones y mucho más.

**Platform as a Service (PaaS)** - plataforma como servicio se refiere al suministro de un entorno bajo demanda para desarrollar, probar, entregar y administrar aplicaciones de software. Está diseñado para crear rápidamente aplicaciones web o móviles, sin preocuparse por configurar o administrar la infraestructura subyacente de servidores, almacenamiento, redes y bases de datos necesarias para el desarrollo.

**Function as a Service (FaaS)** - FaaS añade otra capa de abstracción a la ofrecida por el modelo PaaS. Así los desarrolladores están completamente aislados de todo lo que hay en la pila debajo de su código. En lugar de encargarse de gestionar los servidores virtuales, los contenedores y los tiempos de ejecución de las aplicaciones, cargan bloques de código muy funcionales y los activan ante la ocurrencia de un determinado evento. Las aplicaciones FaaS no consumen recursos IaaS hasta que se produce un evento, lo que reduce las tarifas de pago por uso. Se considera como el futuro del software y las aplicaciones en este sector.

**Software as a Service (SaaS)** - es un método para la distribución de aplicaciones de software a través de Internet según la demanda y típicamente mediante suscripción. SaaS lo ayuda a hospedar y administrar la aplicación de software y a la infraestructura subyacente y gestionar cualquier mantenimiento (actualizaciones de software y parches de seguridad).

**Anything as a Service (XaaS)**

Se refiere a la diversidad creciente de servicios que están disponibles a través de Cloud Computing hoy en día.

Servicios como: *Storage as a Service, Communications as a Service (CaaS), Network as a Service (NaaS), Monitoring as a Service (MaaS), Recovery as a Service (RaaS).* Tambíen servicios emergentes como Marketing as a Service (MaaS), etc.

**Storage as a Service**

El servicio de almacenamiento es una solución de mantenimiento y gestión de datos de los clientes, que el proveedor mantiene en sus servidores y se encarga que esos datos sean accesibles a través de una red, normalmente por internet.

La mayoría de este tipo de servicios están basados en modelo “utility storage”, donde el proveedor pone la capacidad de almacenamiento a disposición del cliente en una base de pago por uso. Normalmente ofrecen una escalabilidad flexible, crecimiento ilimitado y la habilidad de aumentar y disminuir la capacidad de almacenamiento bajo demanda.

Los ejemplos de uso incluyen el servicio de DRaaS (Disaster Recovery as a Service), que consiste básicamente en servicio de copias de seguridad y recuperación de desastres. También se utiliza en colaboración y uso compartido de archivos (FTPs), almacenamiento de datos primarios y etc.

Los típicos ejemplos de ese tipo de servicio son: Amazon S3 (Simple Storage Service), Azure StoreSimple, Google Cloud Storage, etc.

**Network as a Service (NaaS)**

Normalmente las empresas cuando contratan una nube publica, las solicitan por sus capacidades ilimitadas en demanda de almacenamiento y procesamiento. Sin embargo, estas capacidades no importan si no tienen una red adecuada para garantizar la alta disponibilidad y ejecución perfecta de las aplicaciones.

Las redes en Cloud Computing se describen como el acceso a los recursos de redes de un proveedor externo centralizado que utiliza redes de área amplia (Wide Area Networking (WAN)) o tecnologías basadas en Internet.

Las redes en la nube están relacionadas con el concepto de Cloud Computing, donde los recursos de computación centralizados se comparten para los clientes. Es decir, en la nube, sobre todo pública, la red también se puede compartir, así como los recursos informáticos.

Ejemplos de productos de redes ofrecidos de los proveedores líderes:

**Azure**: Traffic Manager, Load Balancer, Azure DNS, VPN Gateway, etc.

**Google:** Load Balancing, Cloud CDN, Cloud DNS, Firewall Rules, Cloud Interconnect, cloud VPN, etc.

**AWS:** Rout 53 DNS, Internet Gateway, Elastic ELB, VPC, Direct Connect, Virtual Private Gateway, etc.

Los servicios mencionados anteriormente se dividen en dos grupos:

- **Servicios no gestionados**

Los servicios no gestionados se administran por el usuario, como configuración de alta disponibilidad de los recursos en la nube, cambios en la carga, los errores y las situaciones cuando los recursos no están disponibles.

En la Figura 4 podemos ver gráficamente de cómo evoluciona el grado de intervención humana para las diferentes tecnologías desde los servicios no gestionados hasta totalmente gestionados.

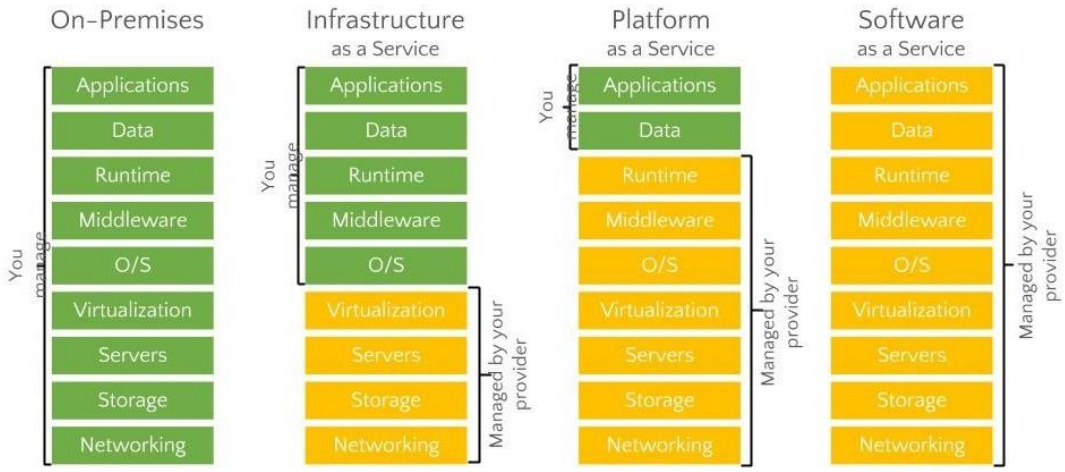


*Figura 4. Los tipos de servicios en Cloud Computing Fuente: James Serra's Blog*

Un ejemplo perfecto de servicio no gestionado es el servicio EC2 de AWS. Un servidor web simple en una instancia de máquina virtual, no se escala de forma automática en caso de mayor carga de tráfico, a menos que se especifique un plan de escalado usando el servicio Auto Scaling. Por tanto, Amazon EC2 es una solución no administrada. Para solucionar todas estas incidencias y configuraciones, el usuario tiene que administrarlos él mismo.

- **Servicios gestionados**

En caso de los servicios gestionados son totalmente al revés. Tienen características de Auto Scaling, tolerancia de fallos, alta disponibilidad y todo está manejado por el proveedor.

Sin embargo, los servicios gestionados aún requieren configuración, pero mucho menos que los no gestionados.

Hay algunos desacuerdos sobre de qué tipo de servicio es mejor, gestionado o no, ya que hay especialistas que piensan que con el servicio no gestionado pueden tener más control del servicio. Al mismo tiempo el servicio gestionado permite a los desarrolladores dedicar más tiempo al desarrollo de sus proyectos y no preocuparse de la administración y configuración de su infraestructura (mantenimiento de las máquinas virtuales, seguridad, alta disponibilidad, etc.).

Un ejemplo de servicio gestionado es Amazon S3, un servicio de almacenamiento de objetos que ofrece escalabilidad, seguridad, disponibilidad de datos y rendimiento. Eso significa que los clientes de todo tipo de tamaño e industria pueden almacenar y proteger sus datos para una gama de casos de uso, como sitios web, aplicaciones móviles, copia de seguridad y restauración, aplicaciones empresariales, dispositivos IoT y análisis de Big Data.

### 2.1.2. Proveedores de Cloud Público

Siguiendo los artículos anuales de Gartner [3], podemos ver claramente cómo evoluciona el mercado de los servicios de infraestructura en la nube. Las empresas líderes siguen dominando el mercado dejando atrás empresas con mucha potencial como Oracle, IBM y Alibaba Cloud (Figura 5).

Como vemos AWS ha sido nombrado líder por noveno año consecutivo, seguido de cerca por Azure de Microsoft. Este es también el primer año, que Google ha entrado en el segmento de Líderes.


*Figura 5.* Cuadrante mágico de Gartner IaaS. Fuente: Gartner Julio 2019

Comparando con los cuadrantes anteriores de los últimos dos años, no ha cambiado mucho el posicionamiento relativo de los líderes. Sin embargo, la mayor diferencia de este año, que resulta algo sorprendente, es que más de la mitad de los proveedores que anteriormente estaban en esta lista han desaparecido, en algún caso pasando tener una cuota del mercado residual (Figura 6).

Si Miramos al cuadrante del año 2017 se ve claramente que han desaparecido bastantes proveedores:

- CenturyLink
- Fujitsu
- Interoute
- Joyent
- Rackspace
- NTT Communications
- Skytap
- Virtustream

Entendemos que la competencia mundial es dura y solo los proveedores premium de hiperescala, que tienen una infraestructura distribuida con la capacidad de escala masiva a una mayor demanda en la computación, son los que tienen la capacidad de sobrevivir en este panorama. Aunque esto no significa que los otros proveedores no tengan relevancia, pero en el mercado global, lamentablemente están fuera.


*Figura 6.* Cuadrante mágico de Gartner IaaS. Junio 2017

Vamos a ver otras fuentes de información sobre el mercado actual, como, por ejemplo, la investigación hecho por Synergy Research Group [4]. Está proporciona datos trimestrales de seguimiento y segmentación del mercado en TI y en mercados relacionados con la nube, incluidos los ingresos de los proveedores por segmento y por región.

Los nuevos datos muestran que el mercado total de los servicios de infraestructura en la nube aumentó un 37% desde el primer trimestre de 2018, superando las tasas de crecimiento alcanzadas en los cinco trimestres anteriores (Figura 7).

AWS sigue dominando la participación en los ingresos del mercado de los servicios de infraestructura y plataforma en la nube. En el siguiente diagrama, solo se muestran los proveedores más importantes en términos de participación de mercado. Estos cuatro proveedores principales dominan aproximadamente tres cuartas partes de todo el mercado.

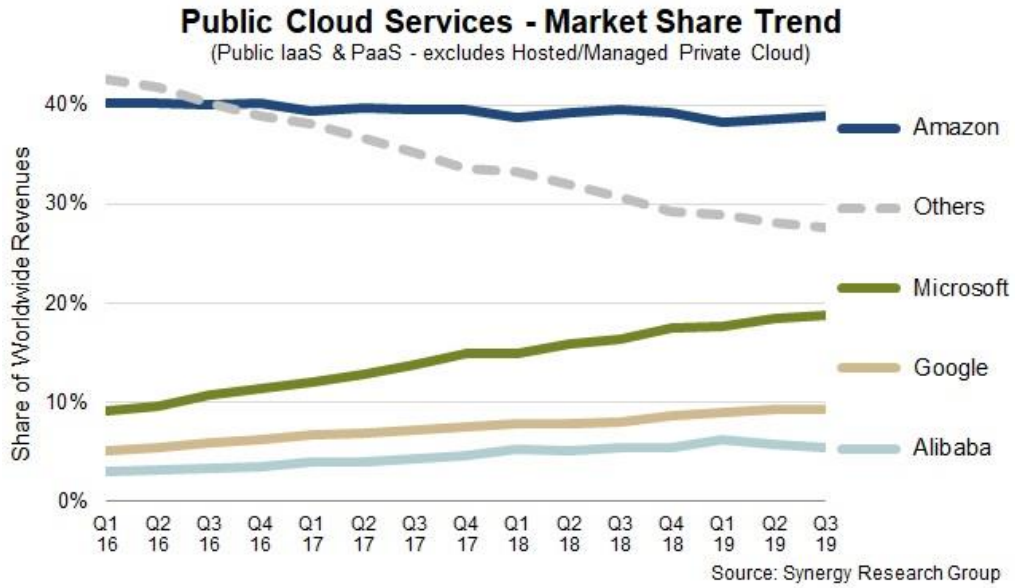


*Figura 7. Servicios de infraestructura en la nube. La tendencia de cuota del mercado. Fuente: Synergy Research Group [4].*

Según Synergy [4] el mercado de los IaaS PaaS público sigue creciendo más rápidamente que los servicios en la nube privado, creciendo hasta un 40%. El mercado de la nube continúa su crecimiento fuertemente en todas las regiones del mundo.

El mercado de IaaS en la nube se está consolidando rápidamente, mientras que los clientes esperan más servicios, características y rentabilidad. Además, la administración, seguridad y administración de las plataformas son cada vez más importantes para el segmento de clientes empresariales. Si un proveedor de servicios en la nube no puede proporcionar estas funciones, no pueden seguir siendo competitivo a nivel mundial y se ve abocado a un nicho de mercado.

Como vemos solo las empresas principales pueden competir en esta amplia gama de características y en mantenerse relevantes para la mayoría de los clientes y los principales casos de uso. Según Gartner, algunos de los proveedores en el cuadrante de nichos tienen grandes aspiraciones de convertirse en jugadores clave, pero el

tiempo dirá si su visión y capacidad de ejecución se desarrollarán y si están dispuestas a desafiar a los tres principales rivales a largo plazo [5].

Después de analizar la situación del mercado de los proveedores de servicios en la nube, veamos los modelos económicos que ofrece el líder del mercado AWS.

### 2.1.3. Modelos Económicos (ejemplificado para AWS)

Con respeto a los modelos de coste de los servicios en la nube, los precios a menudo cambian, más rápido que los propios servicios. Normalmente los precios varían dependiendo a la localización regional, descuentos fuera de las horas pico y uso comprometido.

Hoy en día el precio está variando hacia abajo a medida que crece la competencia. Y para ser competitivos, los proveedores de la nube tienen que adaptarse a estos cambios de precios. Podemos ver variedad de proveedores que ofrecen casi los mismos servicios que AWS, Azure o GCP, aunque son nichos comparando a las tres grandes empresas ya que no llegan a nivel de líder del mercado como el pionero AWS.

Según los representantes de las tres empresas líderes (AWS, Azure, GCP) la influencia del precio de Cloud Computing depende de la competitividad en el mercado. Es decir, cuanta más competencia menos será el costo de los servicios en la nube [6].

Como cada uno de estos tres proveedores tienen diferentes modelos de coste, ejemplificamos el modelo que tiene AWS para sus clientes.

El modelo estándar de pago que AWS ofrece es el modelo de pago por uso. Los modelos de coste de AWS principalmente se dividen en *recursos de cómputo*, *almacenamiento y transferencia de datos* [7]. Los precios varían según el producto, la región y los modelos de precios que tiene el producto.

***Recursos de Computo.*** En caso de uso de recursos de cómputo se paga por segundo, aunque se factura por horas. Empieza la facturación desde el momento cuando se inicia el recurso y la finaliza cuando el recurso se da por terminado. Por ejemplo, según AWS hay cuatro formas de pago las instancias de EC2: *bajo demanda, dedicadas, reservadas e instancias puntuales*. Se puede calcular el precio del modelo de coste a través de AWS Pricing Calculator [8] que permite estimar con precios transparentes el costo de solución de la arquitectura que necesitamos. Para las Instancias reservadas hay elección de tres tipos de pago:

• **Bajo demanda.** No se paga antes de reservar la instancia, aunque los precios son más altos que las otras opciones.

• **Por adelantado parcial.** Se paga una cantidad parcial cuando se reserva la instancia. Los precios en este modelo son menores en comparación del anterior, pero sigue siendo costoso.

• **Completo por adelantado.** Se paga la cantidad entera cuando se reserva la instancia y el precio es menor ya que se paga el pago completo.

Los servicios que pertenecen al modelo de Recursos de Computo son: AWS Lambda, Amazon Lightsail, Elastic Load Balancing, Amazon EC2 Container Registry, Amazon Virtual Private Cloud (VPC) y Amazon EC2.

***Almacenamiento y transferencia de datos.*** Para el uso de los servicios de almacenamiento y transferencia de datos se paga por gigabyte. Este modelo de precio se conoce como *tiered* (escalonado), lo que significa que varios aspectos diferentes del servicio difieren en costo. Por ejemplo, S3 tiene diferentes precios para el almacenamiento, la transferencia y las solicitudes (Figura 8). Igual que los precios de cómputo la región es muy importante para una estimación justa de precio, ya que se varía bastante [9].


*Figura 8.Tipos de Precios para S3. Fuente: Amazon Prices*

En la Tabla 1 podemos ver una lista de servicios principales que ofrece AWS con los precios estimados.

*Tabla 1. Los Precios Estimados de los Productos AWS. Según AWS Pricing Calculator. Ejemplos de propia elaboración.*

| | |
|---|---|
| **Amazon EC2**<br><br>(*Por una instancia básica*)<br><br>Pago por segundo: **0,0000016 $**<br><br>Pago por mes: **4,23 $** | **Lambda**<br><br>*(Sin Free Tier y Concurrencia aprovisionada)*[1]<br><br>**0,20$**/millón solicitudes mensuales<br><br>**0,0000166667$** por cada GB-seg. De duración de cómputo. |
| **S3**<br><br>5GB de uso de almacenamiento estándar S3<br><br>20,000 GET Requests<br><br>20,000 PUT, COPY, POST, o LIST Requests<br><br>Pago estándar mensual **0,24 $** | **AWS KMS**<br><br>2,000,000 de solicitudes y con mínimum 5 clientes gestionados. **11,00$/mes** |
| **Amazon RDS**<br><br>Instancia de db.m5.12xlarge de MySQL<br><br>20 GB de almacenamiento de BBDD: SSD<br><br>20GB de backups con almacenamiento SSD<br><br>Pago mensual **1,923.08 $** | **Amazon SES**<br><br>100 mensajes salientes y entrantes diarios a cualquier destinatario.<br><br>Con el promedio de 10MB de tamaño. **37,41$/mes** |
| **Amazon CloudFront**<br><br>Salida de transferencia de datos primeros 10 TB **0,085$**<br><br>10.000 solicitudes HTTP **0,0090$** y HTTPS **0,0120$** | **Amazon CloudWatch**<br><br>Métricas personalizadas y alarmas de Amazon CloudWatch **0,30$ por métrica/mes**<br><br>Por 100,000 de solicitudes API **4,00 $/mes**<br><br>CloudWatch logs incorporación y almacenamiento **1,26$/mes**<br><br>Eventos de CloudWatch **1$/mes** |

[1] *Esta característica de AWS Lambda la explicaremos en forma más detallada en la sección 2.2.2.1. (pág. 29).*

| AWS Data Transfer | DynamoDB |
| --- | --- |
| 15 GB de transferencia de datos agregados de todos los servicios de AWS. **0,45 $/mes** | *(Ejemplo básico)*<br>Almacenamiento **0,25 $/GB**<br>Lectura y escritura **5,33$/mes**.<br>  |

**Capa gratuita de AWS (free tier)**

La capa gratuita tiene tres tipos de ofertas gratuitas (Tabla 2):

**12 meses gratis.** AWS ofrece para sus nuevos clientes en el momento de registrarse. En esta capa se incluyen; EC2, S3, Amazon RDS y Amazon CloudFront. Por ejemplo, AWS incluye 750 horas para las instancias t2.micro con Windows y Linux al mes durante un año. Para no superar la capa gratuita, se utiliza solo las micro instancias EC2.

**Gratis para siempre.** No expira nunca e incluye los servicios AWS Lambda, AWS KMS, Amazon SES, Amazon CloudWatch y DynamoDB.

**Pruebas.** Esta capa ofrece pruebas a corto plazo que empiezan desde el momento que empieza la prueba. Una vez vencido el modelo se cambia por modelo “pago por uso”.

*Tabla 2. Productos con oferta de la capa gratuita. Propia elaboración.*

| Capa Introductoria Gratuito (12 meses) | Capa gratuita sin caducidad |
|---|---|
| **Amazon EC**<br>  <br>➢ 750 horas gratis de Windows o Linux de instancias t2.micro por mes. | **Lambda**<br>  <br>➢ 1.000.000 de llamadas API recibidas por mes<br>➢ 400.000GB-segundos de tiempo de computación por mes. |
| **S3**<br>  <br>➢ 5GB de uso de almacenamiento estándar S3<br>➢ 20.000 GET requests<br>➢ 2.000 PUT, COPY, POST, o LIST Requests. | **AWS KMS**<br>➢ 20,000 de solicitudes gratis por mes.<br>  |
| **Amazon RDS**<br>  <br>➢ 750 horas de instancia gratuita de db.t2.micro<br>➢ 20 GB de almacenamiento de BBDD: SSD o magnética<br>➢ 20GB de backups con almacenamiento magnético RDS | **Amazon SES**<br>  <br>➢ 62.000 mensajes salientes por mes a cualquier destinatario.<br>➢ 1000 mensajes entrantes por mes. |
| **Amazon CloudFront**<br>➢ Salida de transferencia de datos de 50GB,<br>➢ 2.000.000 solicitudes HTTP y HTTPS de CloudFront<br>  | **Amazon CloudWatch**<br>➢ 10 métricas personalizadas y 10 alarmas de Amazon CloudWatch, 1.000.000 de solicitudes API.<br>➢ 5GB de ingestión de datos de registro.<br>➢ 5GB de archivo de datos de registro.<br>➢ 3 cuadros de mando con hasta 50 métricas cada mes.<br>  |
| **AWS Data Transfer**<br>➢<br>➢ 15 GB de transferencia de datos agregados de todos los servicios de AWS. | **DynamoDB**<br>  <br>➢ 25GB de almacenamiento<br>➢ 25 unidades de capacidad de lectura y escritura.<br>➢ Hasta 200 millones de solicitudes por mes. |

Las tablas fueron creadas en el momento en que se ha realizado este trabajo, por lo tanto los precios y las ofertas de la capa gratuita pueden variar, ya que cambian con bastante frecuencia [10]. De hecho, los precios de EC2 van disminuiendo cada año. Los precios han bajado 67 veces desde que Amazon lanzó EC2 en 2006 [11].

En la capa gratuita entran más servicios como: Amazon SNS, Amazon Lightsail, Amazon GuardDuty, Amazon SageMaker, API Gateway, Amazon CloudFront, Amazon Cognito y algunos más.

Las expectativas de los clientes continuan aumentando significativamente en los ultimos años, y muchos clientes apuestan al uso de IaaS para el ahorro de costos, para tener una agilidad comercial o para tener acceso a las capacidades de infraestructura que no tienen.

El mercado global sigue teniendo a dos líderes claros, Amazon Web Services y Microsoft. El resto del mercado está fragmentado. Aunque, a pesar del dominio completo de dos líderes del mercado, todavía hay muchos de proveedores de servicios que ofrecen IaaS en la nube. Por ejemplo, los proveedores chinos ya han entrado al mercado global, como Alibaba y Tencent, pero aún tienen un éxito limitado fuera del mercado nacional chino.

## 2.2. Serverless Computing

Existen muchas definiciones de Serverless, por lo que definir en pocas palabras este término puede ser difícil. La más técnica y precisa definición de Serverless Computing está en CNCF Serverless Whitepaper, donde se dice [12]: *"Serverless computing refers to the concept of building and running applications that do not require server management. It describes a finer-grained deployment model where applications, bundled as one or more functions, are uploaded to a platform and then executed, scaled, and billed in response to the exact demand needed at the moment."*

En definitiva, es un modelo de ejecución en el que el proveedor de la nube es responsable de la ejecución del código, administrando, aprovisionando y manteniendo servidores para ayudar a los desarrolladores a implementar el código de manera continua (Figura 9). A veces Serverless se define como "NoOps", la evolución del DevOps.

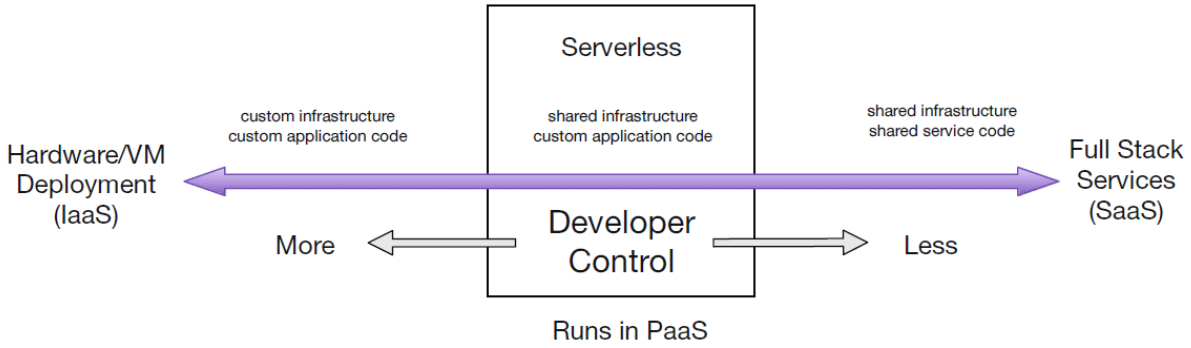


*Figura 9. Control de desarrollador y Serverless Computing. Fuente: I. Baldini et al., "Serverless Computing: Current Trends and Open Problems," 2017, pp. 1–20* [13].

El objetivo aquí es a pesar de que el modelo se llama Serverless, sí que hay servidores involucrados, pero queremos pensar en nuestro código y no preocuparnos por los detalles de la infraestructura en la que se ha implementado como el balanceador de carga, el escalado automático, la alta disponibilidad, etc. Solo estamos enfocados en la codificación, elevándolos y dejando que el servicio se preocupe por el aprovisionamiento automatizado de recursos de cómputo.

Es ahí donde queremos llegar y en particular, en esta sección hablaremos de las arquitecturas de Serverless, tipos de servicios existentes y los modelos de coste de los proveedores que ofrecen.

En la Figura 10 se muestra la creciente popularidad del término de búsqueda "Serverless" en los últimos 5 años según Google Trends, lo que indica cómo ha crecido el interés hacia el Serverless Computing en la comunidad de desarrollo durante este tiempo.

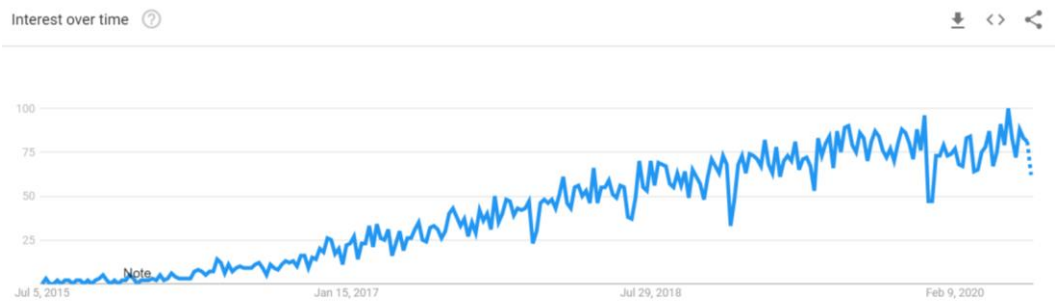


*Figura 10. Popularidad del término "Serverless" en los últimos 5 años según Google Trends.*

Sin embargo, según el articulo de investigacion "Serverless Computing: Current Trends and Open Problems" [13], el metodo del Serverless no es completamente nuevo. Ha surgido tras los recientes avances y la adopción de la máquina virtual (VM)

y luego las tecnologías de contenedores. Cada paso en las capas de abstracción condujo a unidades de cálculo más ligeros en términos de consumo de recursos, costo y velocidad de desarrollo e implementación.

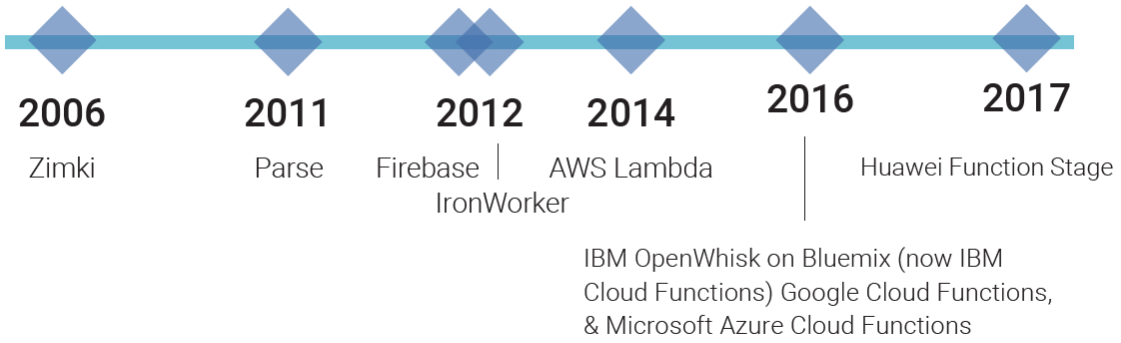


*Figura 11. Una breve historia de la tecnología Serverless. [8]*

Mientras la idea de "pago por uso", el modelo se rastrea a 2006 y una plataforma llamada Zimki, uno de los primeros usos del término Serverless es de Iron.io en 2012 con su producto IronWorker, una plataforma de trabajo bajo demanda distribuida, basada en contenedores [12], tal y como se muestra en la Figura 11.

Primero había servicios BaaS (Backend-as-a-Service) que son servicios basadas en API de terceros que reemplazan subconjuntos básicos de funcionalidad en una aplicación. Esas APIs se proveían como servicio auto escalable y de forma transparente, de los cuales eran Parse y Firebase (en los años 2011 y 2012), que luego fueron adquiridos por Google y Facebook [12]. La popularización de Serverless empezó cuando Amazon presento AWS Lambda [14] en la sesión de re:Invent en 2014 [15].

Después siguieron los otros proveedores en 2016; Google con su Cloud Functions [16], Microsoft con Azure Functions [17] y otros más. También existen diferentes Serverless frameworks como SAM [18], Zappa [19], Sparta [20] y Serverless.com [21].

El framework Serverless tiene soporte para varias plataformas como Azure Functions, Cloud Functions, AWS Lambda y WebTasks. Es multi lenguaje, que se puede escribir funciones en Python, NodeJS, Java, Go, Scala, C#, etc.

En general los servicios Serverless soportan una amplia variedad de lenguajes de programación, incluidos Javascript, Java, Python, Go, C#, y Swift. Las plataformas mencionadas anteriormente admiten más de un lenguaje de programación.

Los beneficios principales que ofrece el modelo Serverless es:

- **Zero Server Ops:** Serverless al eliminar la sobrecarga involucrada en el mantenimiento de los recursos del servidor cambia el modelo de costo para la ejecución de aplicaciones. La administración y mantenimiento de servidores,

MVs y contenedores es un gasto significativo para las empresas cuando se incluyen personal, herramientas, formación y tiempo. Mientras con Serverless estos tipos de gastos se reducen enormemente, ya que es un modelo completamente gestionado que controla la infraestructura subyacente de la arquitectura, suponiendo un ahorro para ciertos tipos de cargas de trabajo que no requieran un uso intensivo y continuado de la infraestructura.
Serverless puede escalar de manera instantánea y precisa para gestionar cada solicitud entrante. El escalado ocurre automáticamente sin intervención del desarrollador. Al finalizar el procesamiento reduce automáticamente los recursos de cómputo para que no haya capacidad inactiva.

- **Sin costo de cómputo cuando está inactivo:** Uno de los mayores beneficios de los productos Serverless para los usuarios, es que no hay costos en caso de la capacidad inactiva. Es decir, no hay cargo cuando el código no se está ejecutando o no está realizando un trabajo significativo.

Para entender cómo son las aplicaciones Serverless vemos un pequeño ejemplo comparativo de la arquitectura Serverless y arquitectura "on-premises" extraída del libro "What is Serverless?" [22].

Con unos requisitos como interfaz de usuario, gestión de usuario y autenticación, lógica de juego y tablas de clasificación que requiere una aplicación de móvil, para la arquitectura on-premises necesitaríamos, tal como se ve en la Figura 12:

- ✓ Una aplicación móvil para iOS/Android,
- ✓ Un backend que se ejecuta en un servidor,
- ✓ Una base de datos relacional, como MySQL.

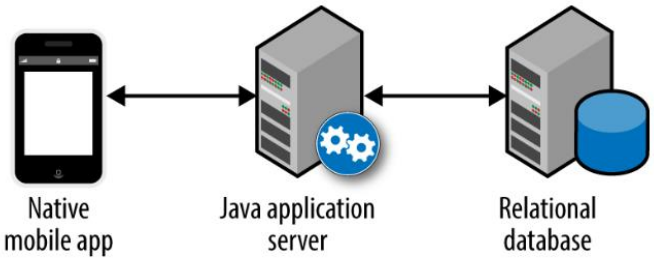


*Figura 12. Arquitectura on-premises para un juego de móvil* [22].

Mientras tanto, con una arquitectura Serverless podemos eliminar muchos de los desafíos relacionados con la administración de la infraestructura y los sistemas subyacentes que la aplicación utiliza (Figura 13).

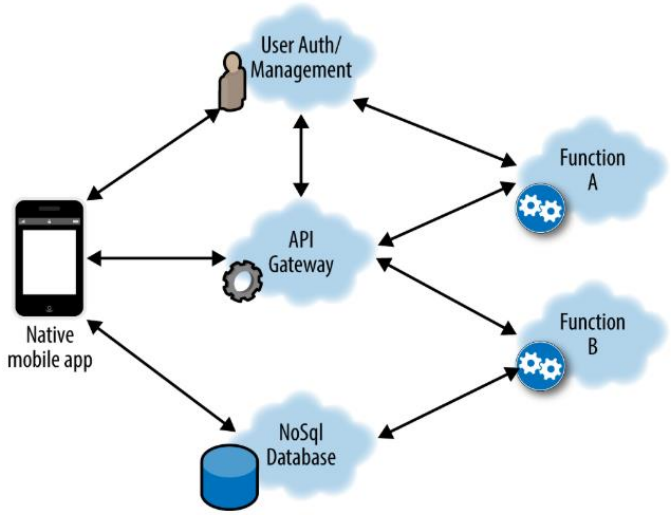


*Figura 13. arquitectura Serverless para un juego de móvil* [22].

Gracias al auto escalado, la alta disponibilidad y la seguridad, no tenemos que preocuparnos ni por problemas de configuración de firewall, ni por la caída de base de datos o compra de más servidores en caso del aumento de los usuarios, que es un "cuello de botella".

En resumen, con la arquitectura Serverless reduciríamos:

- el coste laboral, ya que no necesitamos el mantenimiento de los servidores,
- el coste de los recursos,
- mayor flexibilidad de escalado,
- riesgo reducido,

y el tiempo de entrega de la aplicación será menor.

### 2.2.1. Diferencias entre Serverless y FaaS

Aunque el paradigma de Serverless y FaaS se parezcan, sin embargo, no son lo mismo. Según Paul Johnston [23] (cofundador de la comunidad ServerlessDays y ex AWS Serverless Senior Developer Advocate), una solución sin Serverless se construye usando FaaS. El solo uso de funciones no hace que algo sea automáticamente Serverless.

Citando al mismo Johnston [23]: *"El problema es que, si bien DevOps nos dice que la automatización es importante, las soluciones Serverless nos han dado una dimensión que a menudo no se considera, y es que la infraestructura Serverless es parte de la solución, no simplemente en qué se ejecutan las soluciones."*

FaaS recibe la mayor parte de la atención en estos días, en gran parte porque es el aspecto más transformador de Serverless. Uno de los servicios utilizados de FaaS es AWS Lambda, actualmente número uno en el mercado de Serverless. Existen también

servicios de almacenamiento Serverless, incluido los servicios de almacenamiento en bloque, como AWS S3, y las ofertas de base de datos como servicio (DBaaS), como Amazon DynamoDB, Firebase de Google e IBM Cloudant. Estos son Serverless en el sentido de que no tiene que configurar una matriz redundante de RAIDs o decidir cómo equilibrar la carga de su clúster de BD.

BaaS y FaaS están relacionados en sus atributos operativo y se usan con frecuencia juntos. Lo único que muchas veces se toman por mismo Serverless y FaaS, es que los dos tienen el modelo de "pago por uso", es decir, se paga cuando se utiliza o está en ejecución el código del desarrollador. Y los dos están libres de construcción y mantenimiento de la infraestructura de back-end.

Otros servicios como AWS EC2 también tiene el modelo de pago por uso, aunque realmente se trata de un "pago por despliegue". No importa si se usa o no, se incurre en los gastos. Sin embargo, FaaS realmente ofrece un pago por uso real, ya que se incurre en el gasto sólo cuando está en estado de ejecución.

Los productos FaaS que se utilizan más en las soluciones de desarrollo hoy en día son AWS Lambda, Google Cloud Functions, Microsoft Azure Functions y las opciones de Oracle e IBM (Oracle Functions [24] y IBM Cloud Functions [25]), que son de código abierto.

Terminando la idea de la diferencia entre Serverless y FaaS con una definición más completa y técnica podemos decir, que *FaaS* es como un modelo de computación basada en eventos donde los desarrolladores ejecutan y administran el código de la aplicación con funciones que se activan por eventos o solicitudes HTTP. Mientras la arquitectura *Serverless* es un diseño de aplicación que incorpora servicios de Backend as a Service (BaaS) de terceros y/o que incluye código personalizado ejecutado en entornos administrados en una plataforma FaaS. En muchos sentidos, la arquitectura Serverless se parece a otros diseños de aplicaciones centrados en eventos y microservicios [26].

Para tomar una solución correcta en un proyecto de desarrollo es un paso imprescindible entender que Serverless y FaaS no son lo mismo.

Según Peter Sbarski (vicepresidente de ingeniería en la escuela de Cloud Computing "A Cloud Guru") [27], en ser Serverless no se trata solo de ejecutar código en Lambda, también se trata de utilizar servicios y productos de los terceros. La combinación de funciones escalables en la nube, servicios fiables de terceros y arquitecturas Serverless y patrones, es el siguiente paso en la evolución del Cloud Computing.

### 2.2.2. Servicios Disponibles

En el mercado actual existen múltiples frameworks y servicios Serverless que los proveedores hoy en día ofrecen. Como anteriormente destacamos los tres principales proveedores líderes de Cloud Infrastructure as a Service (AWS, GCP y Microsoft

Azure), veremos sus productos FaaS que ofrecen los mismos, que serían: AWS Lambda, Google Cloud Functions y Azure Functions.

#### 2.2.2.1. AWS Lambda

Es un servicio popular que hoy en día es uno de los pilares de las funciones Serverless. Permite ejecutar código sin aprovisionar ni administrar servidores. Se paga solo por el tiempo de cálculo que consume, no hay cargo cuando el código no se está ejecutando. Con Lambda, se puede ejecutar código para prácticamente cualquier tipo de aplicación o servicio backend [14].

Según el cofundador de Symphonia Serverless John Chapin [22], es una de las plataformas FaaS más maduros y estables disponibles en el mercado. En el principio ha empezado con Node.js, hoy en día soporta Go, Java, Python, Ruby y C#/PowerShell [28] (con apoyo de .NET Core. Una plataforma de desarrollo, dedicado para los sistemas operativos Windows). Está integrada con más de una docena de servicios de AWS (Figura 14), por lo tanto, los desarrolladores eligen AWS Lambda, por su flexibilidad y la potencia que blinda.


*Figura 14. Fuentes de eventos que generan AWS Lambda. Elaboración propia con draw.io*

Lambda ejecuta el código en respuesta a eventos. Su función se invoca cuando estas fuentes de eventos detectan eventos. Por ejemplo, si queremos crear una miniatura de imagen (Thumbnail) para cada archivo que se carga en un bucket de

almacenamiento S3 (Figura 15). Para ello se crea un código de la función Lambda que lee el objeto de imagen del bucket de origen (Source Bucket) y crea una imagen en miniatura y almacena en el bucket de destino (Target Bucket) [29].

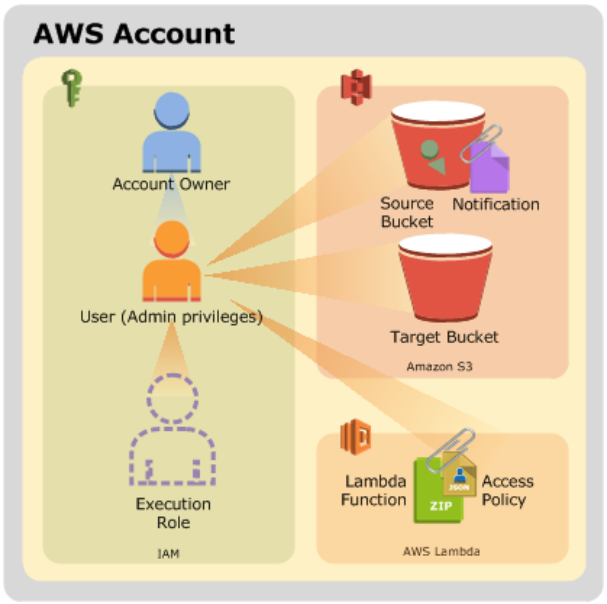


*Figura 15. Diagrama de creación de miniatura de imagen usando AWS Lambda con Amazon S3. Fuente: AWS.*

AWS Lambda es a menudo el centro de la arquitectura Serverless del AWS, que permite ejecutar código sin aprovisionar ni administrar servidores. Se basa en una arquitectura que activa el código en respuesta a eventos y se ejecuta en paralelo. Puede ejecutar código para casi cualquier tipo de aplicación o servicio backend. Lambda se encarga de ejecutar y escalar el código con alta disponibilidad. Lo único que hay que hacer, es subir el código en uno de los lenguajes que admite AWS Lambda (Python, Java, Node.js etc.). La facturación de AWS Lambda es por bloques de cada 100 ms de ejecución del código y por el número de veces que se activa el código [14]. Se paga solo por el tiempo de cómputo que consume por la ejecución, mientras que con EC2 a pesar del pago por segundos (aunque los precios se muestran por horas), se paga por estado inactivo también (pay for idle), pero únicamente para los volúmenes EBS de datos vinculados a la instancia.

***Las ventajas:***

- ***Costes operacionales reducidos:*** La implementación en la infraestructura de AWS puede ser muy rentable comparado con el salario de los ingenieros backend y los costes de mantenimiento de infraestructura.
- ***Desarrollo más rápido****:* se invierte menos tiempo en problemas operacionales. Lo que permite tener menor Time-To-Market (plazo de lanzamiento).

- ***Reducido coste para escalado masivo**:* Los eventos de Lambda se amplían para tomar todo el tráfico entrante. Sin usar ni una instancia EC2.

***Las desventajas:***

- ***Cold start**:* Debido a la estructura bajo demanda, en algunos entornos de ejecución Serverless (runtime) pueden tener problemas de rendimiento, ya que se eliminan los recursos aprovisionados para la ejecución de una función cuando está inactivo.
  El "cold start" ocurre cuando se ejecuta una función inactiva, ya que después de un período de inactividad el proveedor elimina dichos recursos, provocando que aumente el tiempo de ejecución. Para este problema ya existen varias formas de mitigarlo, uno de los cuales es Provisioned Concurrency [30], (explicación más detallada ver en la parte Características).
- ***No apropiado para ejecuciones largas**:* Cuando las funciones requieren la ejecución de código computacionalmente intensivo o tienen flujos de trabajo complicados.
- ***Limitaciones**:* Otra de las desventajas son los límites de AWS Lambda que se aplican a la configuración implementación y la ejecución de la función. De los principales son [31]:
  - Asignación de memoria – máximo 3.008 MB en incremento de 64 MB.
  - Espacio del disco efímero (/tmp) – 512 MB.
  - Duración de ejecución por petición – 15 minutos.
    (Para una información más detallada sobre los límites de los tres proveedores ver la Tabla 3)
- ***Vendor lock-in**:* Dificultad de migrar una aplicación de un proveedor a otro. Eligiendo el FaaS de AWS, dependes de él.

En primer lugar, hablemos sobre los bloques importantes en el concepto Serverless que se construye con AWS Lambda.

API Gateway [32] – permite configurar puntos finales (endpoints) con Lambda. Es una "puerta de entrada" para el acceso de las aplicaciones a los datos de comunicación, bidireccional en tiempo real.

DynamoDB [33] – para el alojamiento de datos (BBDD).

CloudFormation [34] – es infraestructura como código, para modelar y aprovisionar recursos de AWS en el entorno de la nube.

AWS SAM (Serverless Application Model) [18] – otra de las novedades de AWS creada especialmente para el desarrollo de las aplicaciones Serverless. Según la documentación de AWS, es una combinación de funciones Lambda, fuentes de eventos y otros recursos que trabajan juntos para realizar tareas. Ya que como

citamos anteriormente que una aplicación Serverless no es solo una función Lambda, sino un conjunto de recursos adicionales.

Tiene siguientes características:

- ✓ Extensión CloudFormation optimizada para Serverless.
- ✓ Nuevos tipos de recursos Serverless: APIs, tablas y funciones.
- ✓ Las plantillas de SAM son las mismas que de las CloudFormation.
- ✓ Es de código abierto bajo la licencia Apache 2.0

Una alternativa a SAM que es muy utilizado, aunque no pertenece al core de servicios de AWS es Serverless framework [21] – permite administrar Lambda, API Gateway y CloudFormation a través del código y AWS CLI en lugar de GUI (Graphical User Interface) como a través del navegador web (AWS Management Console).

***Modelo de ejecución.***

Existen una serie de modelos [35] de ejecución de Lambda, diferentes maneras en que podemos invocar la función. Puede elegir entre invocación síncrona y asíncrona.

*Síncrona* (push) – con API Gateway configurado para la función Lambda, realizamos una solicitud a la API e inmediatamente recibimos una respuesta con los resultados de lo que se ha ejecutado dentro de la función visualizando los logs en CloudWatch Logs.

*Asíncrona* (evento) – el evento se pone en cola para su procesamiento y devuelve una respuesta inmediata. Según la documentación [35] de AWS Lambda, para las invocaciones asíncronas, Lambda maneja reintentos si la función devuelve un error o está limitada al superar el límite de ejecuciones síncronas concurrentes.

***Características*:**

- ✓ *Bring your Own Code:* soporte de diferentes lenguajes de programación.
- ✓ *Integración* a otros servicios de AWS.
- ✓ *Stateless:* funciones sin estado, que no son sensibles al estado de la computación.
- ✓ *Monitorizacion:* métricas para peticiones (CloudWatch metrics), errores, *cloudwatch logs*.
- ✓ *Cold Start*: Latencias añadidas en primeras ejecuciones de la función lambda.
- ✓ *Throttling:* Cuando la función lambda supera el número de ejecuciones concurrentes o las solicitudes llegan más rápido de lo que puede escalar, Lambda para la función con un error de limitación (error 429) [36].
- ✓ *Provisioned Concurrency* [30]: Es la nueva característica para AWS Lambda que permite mantener la función inicializada y preparada (“caliente”) para poder responder en milisegundos de dos dígitos. Según AWS es ideal para implementar en los backends de los servicios como web y móvil o

microservicios sensibles a la latencia (p.ej. microservicios para el sistema de comercio financiero) [30]. Cuando se activa la Concurrencia Aprovisionada, Lambda inicializa el numero solicitado de entornos de ejecución. De esta manera las funciones están preparadas para responder a las invocaciones.

En la conferencia de re:invent 2018 [37], también fueron anunciado dos novedades relacionados a su arquitectura Serverless: Lambda Layers y Runtime API.

*Lambda Layers* es una forma de gestionar el código y las dependencias de forma centralizada. Una capa es un archivo de ZIP que contiene librerías u otras dependencias con que se puede usar en la función sin tener que incluirlas al paquete de implementación. Con Layers se resuelven algunos problemas que solían tener los desarrolladores antes, como para la implementación de dependencias a más de una función.

*Lambda Runtime* API ofrece una nueva interfaz para usar cualquier lenguaje de programación o una versión de lenguaje particular para desarrollar funciones.

#### 2.2.2.2. Google Cloud Functions

Es una plataforma de cómputo Serverless basada en eventos que se puede crear usando Java Script, Python 3 o Go. Es uno de los conjuntos de servicios Serverless que ofrece Google Cloud Platform.

La clave diferente de Google Cloud Functions es que conecta y extiende servicios como Google Assistant, Firebase y servicios de terceros (GitHub, Slack), como bloques para construir una aplicación Serverless. El aprovisionamiento de recursos ocurre automáticamente y el precio se factura en bloques de 100 milisegundos de ejecución de la función. Es perfecto para los casos de uso como procesamiento de datos/ETL, APIs ligeros, Webhooks o IoT. Igual que AWS Lambda y Azure Functions se puede desplegar las funciones con el Serverless Framework [21] que es de código abierto y herramienta fácil de desarrollo y permite desplegar colecciones de recursos descritas mediante lenguaje YAML + CLI.

Al servicio Serverless se incluyen los productos de Google Cloud como Pub/Sub, Cloud Tasks, Cloud Scheduler, Firestore, Cloud ML, Vision API y BigQuery.

Una de las ventajas de Cloud Functions es la tecnología open source FaaS para ejecutar funciones como entornos Cloud Run, Cloud Run para Anthos y entornos Serverless basados en KNative [16].

Los casos de uso común son:

- *Aplicaciones backend:* Integración con servicios terceros y APIs, Serverless beck-end para los móviles y IoT.
- *Procesamiento de datos en tiempo real:* procesamiento de archivos y flujo (stream) como se muestra en la Figura 16.

- *Aplicaciones inteligentes:* asistente virtual y conversación, análisis de los sentimientos, video e imagen.

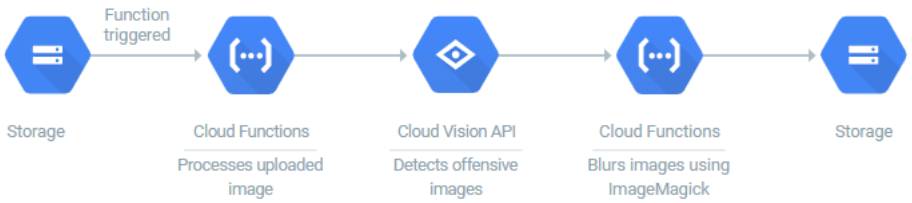


*Figura 16. Procesamiento de flujo de imágenes en tiempo real. Fuente: Google Cloud*

#### 2.2.2.3. Azure Cloud Functions

Según la página web Datamation [38], Azure Functions es una solución ideal para los desarrolladores que utilizan Visual Studio Code, que proporciona un camino simple para el desarrollo e implementación de funciones Serverless. Las funciones se pueden ejecutar en dos entornos: Windows y Linux. Ofrece la integración con una serie de servicios de Azure y servicios de terceros (como Azure Event Grid, Azure Storage, Azure CosmosDB, Azure Service Bus, Logic App, API Management, etc.) con los que ejecutan fragmentos de código. El servicio soporta una variedad de lenguajes de programación como C#, F#, Node.js, Python, PHP o Java. También admite opciones de secuencias de comandos como Bash, Batch y Powershell.

Azure Functions tiene las siguientes características [39]:

- Paquete de servicios para el desarrollo de aplicaciones Serverless,
- Admite dependencias como NuGet y NPM con acceso a sus librerías,
- Seguridad integrada: proveedores OAuth (como Azure Active Directory, Facebook, Google, Twitter y la cuenta de Microsoft),
- Integración simplificada con los servicios de Azure y ofertas SaaS,
- Desarrollo flexible con la implementación mediante GitHub, Azure DevOps Services
- Arquitectura Serverless con estado: aplicaciones Serverless con Durable Functions (una extensión de Azure Functions que permite crear flujos de trabajo basados en código administrando estados, puntos de control y reinicios [40]),
- El entorno de ejecución es de código abierto.

La supervisión de las funciones está integrada vía Azure Application Insights que se implementa en casos de solución de problemas de rendimiento.

La función de Azure Functions se ejecuta el código por los triggers. Al igual que AWS, Azure Functions ofrece triggers dinámicos y configurables que pueden usarse para invocar las funciones. Hay varias maneras de desencadenar Azure Functions:

- CRON expresiones,
- Cambios en los contenedores de Azure Storage Blob,
- Cambios en Azure Queues,
- Mensajes del Service Bus,
- HTTP triggers.

Una sola función debe tener solo un trigger.

Según A. Kumar, el entorno Azure Functions es una perfecta plataforma para procesamiento de datos, que ofrece capacidades para integrar sistemas a través de enlaces de entrada y salida. También se adapta a casos de uso de ingestión de Big Data como IoT y Azure Data Factory tal como se muestra en la Figura 17 [40].

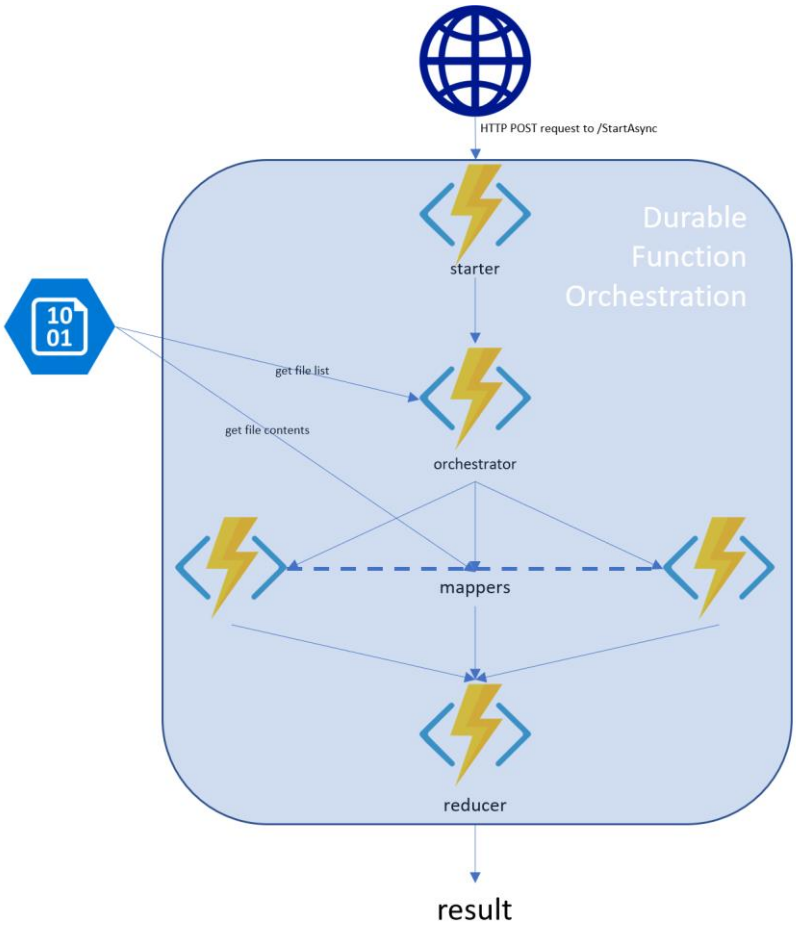


*Figura 17. Escenario de Procesamiento de Big Data: Velocidad media de los viajes realizados por los taxis de Nueva York por día en 2017. Fuente: Microsoft Azure*

En la Tabla 3 podemos ver la comparación de los precios y las limitaciones de cada servicio que ofrecen los tres proveedores, donde se puede ver las ventajas y desventajas de cada uno.

*Tabla 3. Los precios y limitaciones de cada proveedor*

| Proveedores | AWS Lambda | Azure Functions | Google Functions |
|---|---|---|---|
| **Precios** | 1 millón peticiones por mes gratis<br>—<br>400.000GB-segundos de tiempo de computación por mes<br>—<br>Después 0.20$/1millión peticiones<br>[14] | 0.000016$/GB, 400,000 GBs/mes gratis<br>—<br>0.20$/millón ejecuciones,<br>—<br>1 millón ejecuciones/mes gratis<br>[16] | 0.40$/millón invocaciones con 2 millón invocaciones gratis<br>—<br>0.0000025$/GB-seg. con 400,00 GB-seg. /mes gratis<br>—<br>0.0000100$/GHz-seg. con 200,000 GHz-seg. /mes gratis<br>[17] |
| **Limitaciones** | Permite entre 1000 y 3000 ejecuciones concurrentes según la región<br>—<br>Rango de asignación de memoria Min. 128 MB / Max. 3008 MB Almacenamiento de directorio /tmp: 512 MB<br>—<br>Duración de ejecución por petición:15 minutos<br>(más info.) | Permite máx. 10 ejecuciones concurrentes por función<br>—<br>Plan premium - 14GB de RAM, con instancias calientes<br>—<br>Sin límites en tiempo de ejecución<br>(más info.) | Numero de funciones: 1000<br>—<br>Máx. duración de función: 540 seg.<br>—<br>Llamadas de función por seg.: 1,000,000 por 100 seg.<br>(más info.) |

# 3. Arquitecturas de Aplicaciones Serverless

Tras haber visto las tecnologías de Cloud Computing y sobre todo el modelo de computación Serverless, a continuación, vamos a centrarnos en nuestro objetivo, que es un análisis de las arquitecturas Serverless más destacadas y sus modelos económicos. Finalmente completaremos el capítulo con el caso práctico que hemos hablado anteriormente en la Introducción .

## 3.1. Tipos de Arquitecturas Serverless y sus modelos económicos

La arquitectura Serverless se puede implementar en muchos casos y por eso, en este párrafo recopilamos los tipos de arquitecturas más destacados que existen en “cloud-native computing”.

Según el Whitepaper de CNCF la solución Serverless debe considerarse una opción cuando la carga de trabajo es [12]:

- ✓ Ejecución lógica: cambios en la base de datos (insertar, actualizar, activar, eliminar)
- ✓ Realización de analítica en mensajes de entrada del sensor IoT (mensajes de protocolo MQTT);
- ✓ Manejo del procesamiento de flujo (analítica o modificación de datos en tiempo real);
- ✓ Administración de extracción única, transformación y carga de trabajos que requieren mucho procesamiento por un corto tiempo (ETL);
- ✓ Proporción de computación cognitiva vía una interfaz de chatbot (asíncrono, pero correlacionado);
- ✓ Programación de tareas realizadas por poco tiempo (invocaciones de Cron o Batch);
- ✓ Modelos de Machine Learning e Inteligencia Artificial (recuperación uno o más de elementos de datos y comparándolos con un modelo de datos previamente aprendido para identificar texto, caras, anomalías, etc.);
- ✓ Tuberías de integración continua que proporcionan recursos para crear trabajos a demanda, en lugar de mantener un grupo de hosts esclavos en espera a la consignación de trabajo.

Muchas empresas como iRobot, Netflix y NY Times ya han adoptado la arquitectura Serverless en sus soluciones, ahorrando muchos recursos y esfuerzo para el desarrollo backend.

En este trabajo vamos a enfocar más al proveedor de la arquitectura Serverless AWS, ya que se considera uno de los primeros en esta área, que tiene una gran lista de productos Serverless más estables y maduros.

A continuación, veremos unos casos de uso más comunes de la arquitectura Serverless, desplegando cada uno de ellos con ejemplos prácticos y comparando con una arquitectura tradicional basada en instancias de EC2.

### *Aplicaciones web*

Es uno de los usos más comunes de la arquitectura Serverless.

Una aplicación web presenta una interfaz de usuario basada en HTML, dedicada a la prestación de servicios públicos en Internet. En el siguiente ejemplo la arquitectura consiste en uso de aplicaciones como AWS Lambda, Amazon API Gateway, Amazon Cognito, DynamoDB y Amplify Console (Figura 18).

La arquitectura tiene cuatro capas [41]:

- AWS Amplify proporciona un despliegue continuo y alojamiento de los recursos web estáticos incluidos HTML, CSS, JavaScript. (La alternativa sería el despliegue de la web en el Amazon S3.)
- Amazon Cognito administra y autentica a los usuarios.
- Un código JavaScript ejecutado en el navegador envía y recibe datos de una API de backend pública creada con Lambda y API Gateway.
- DynamoDB: BDD no relacional que almacena los datos por la función Lambda de la API.

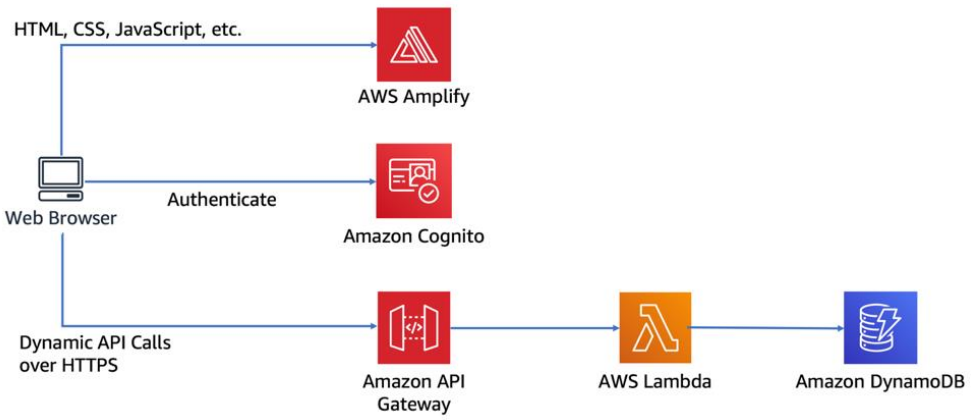


*Figura 18. Arquitectura de una aplicación web Serverless. Fuente: AWS Serverless web application*

Ahora veremos cómo sería una aplicación web con la arquitectura basada en EC2 según AWS [42]. En la Figura 19 se muestra un ejemplo creada por AWS de una arquitectura clásica para las aplicaciones web y cómo se podría diseñarla.

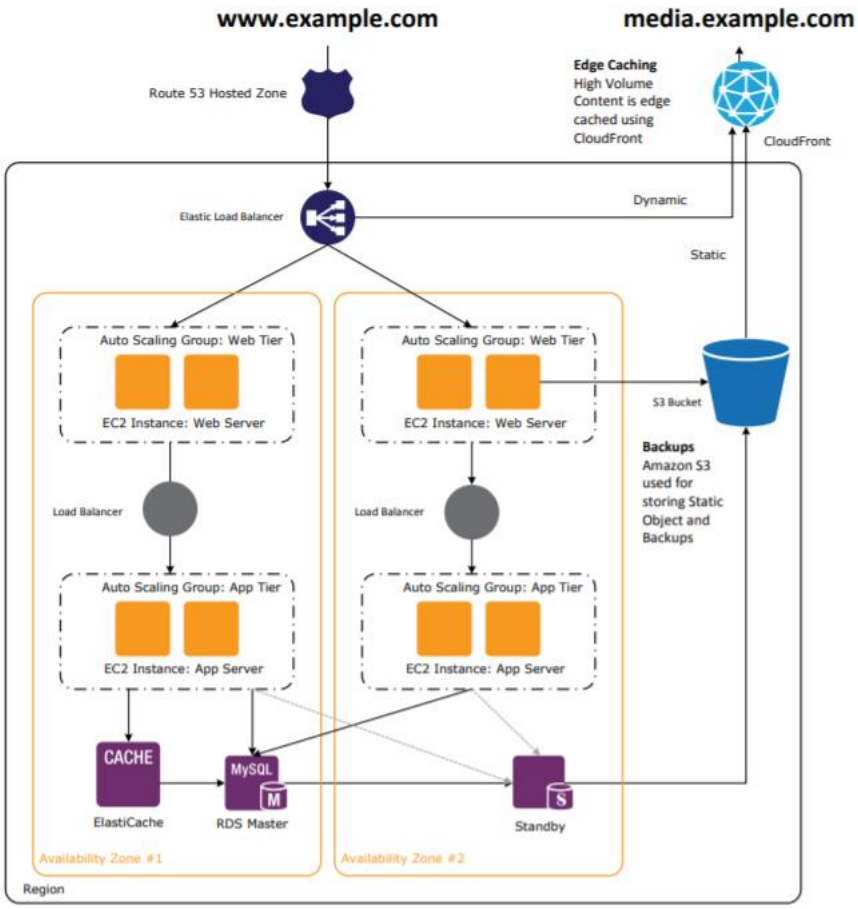


*Figura 19. Una arquitectura tradicional para una aplicación web en AWS. Fuente: AWS.*

La infraestructura tendría los siguientes componentes principales:

- *Route 53* – con la zona alojada, proporcionaría servicios de DNS para simplificar la administración de dominio.
- *Elastic Load Balancer (ELB)* – para distribuir el tráfico a los grupos de Auto Scaling de los servidores web y servidores de la aplicación en las instancias EC2, para distribuir el tráfico en el clúster de servidor de la aplicación.
- *Los grupos de Auto Scaling* – Grupos de instancias EC2 de la capa web (que manejan solicitudes HTTP) y la capa aplicación (que ejecutan la aplicación real).
- *Exterior Firewall* – en cada instancia de servidor web a través de grupos de seguridad. Que permitiría un acceso para cualquier host solo a través del protocolo TCP (HTTP y HTTPS).
- *Backend Firewall* – en las instancias de backend que significa que tendría un grupo de seguridad del clúster del servidor de aplicaciones. También podría permitir el acceso desde su subred de la organización a través de SSH (Secure Shell – un protocolo de administración remota) para la administración directa del host.

- ElasticCache – podría reducir la carga del servicio y mejorar la latencia y la escalabilidad de bases de datos con aplicaciones de cachés en memoria.
- Bases de datos MySQL RDS – según Amazon [42], los bases de datos relacionales (RDS) con implementación de Multi-AZ (Multi Availability Zone) se crearía una arquitectura de alta disponibilidad, con sus réplicas.
- También Amazon recomienda usar volúmenes de Amazon Elastic Block Storage (Amazon EBS), similar a almacenamiento conectado a la red, que tendría tolerancia a fallos [42].

### *Big Data*

Dado que el concepto de Big Data consiste en las enormes dimensiones de datos y dificultades que nos enfrentamos para el procesamiento de datos masivos, para ello se crea procesos ETL (extracción, transformación y carga), utilizando las funciones Lambda. Y para el análisis, AWS dispone de una variedad de herramientas (Figura 20). Implementando la arquitectura Serverless en Big Data permite estar enfocado en la analítica y no en la infraestructura.

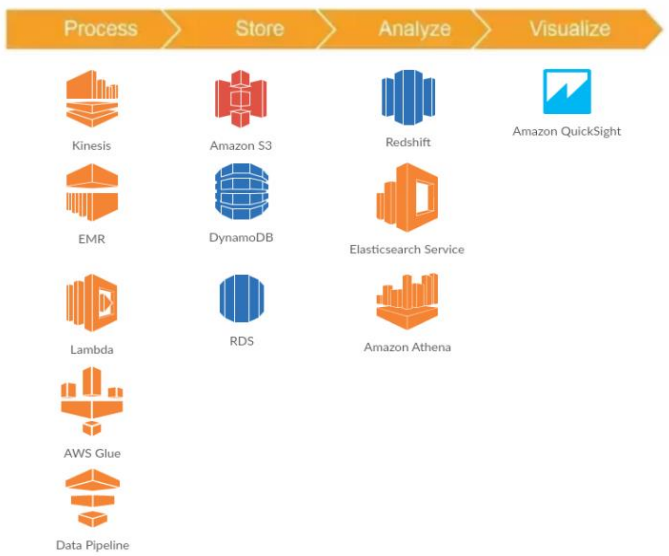


*Figura 20. Servicios de análisis de datos en AWS. Fuente: Elaboración propia con Creately* [43]

AWS dispone de los siguientes servicios para implementación en Big Data [43]:

*Amazon Athena:* Servicio de consulta Serverless que permite analizar fácilmente los datos en Amazon S3 con SQL estándar.

*Amazon EMR*: Proporciona un framework Hadoop administrado para procesar grandes volúmenes de datos de forma rápida.

*Amazon Elasticsearch Service:* Facilita la implementación, la operación y el escalado de Elasticsearch en AWS.

*Amazon Kinesis:* Recopilación y análisis de datos de streaming en tiempo real.

*Amazon Redshift:* Almacén de datos rápido y totalmente administrado a escala de petabytes.

*AWS Glue:* Servicio Serverless que prepara y carga datos a almacenes de datos.

*AWS Data Pipeline:* Organización del flujo de trabajo de datos.

*Amazon QuickSight:* Un servicio de Business Inteligence.

Respecto a los servicios mencionados anteriormente se puede construir diferentes tipos de escenarios de arquitectura Serverless en Big Data. Por ejemplo:

- ✓ visualización, canalización y migración de datos masivos,
- ✓ creación y automatización de un Data Lake,
- ✓ mecanismo totalmente administrado para actualizaciones en lagos de datos,
- ✓ ingestión de datos automáticamente,
- ✓ procesamiento de datos en tiempo real y mucho más…

*Data lake* es un concepto de depósito centralizado donde puede almacenar todos los datos no estructurados y estructurados, como están, a cualquier escala. AWS S3 se considera como la base de cualquier lago de datos, que permite centralizar conjuntos de datos. Es simple y fácil de manejar. Es el almacén de objetos fundamental para que todos los datos sean catalogados, analizados y transformados en datos interesantes.

Veremos un caso de uso muy reciente, relacionado con lagos de datos creada con las herramientas Serverless: AWS ha creado un lago de datos de información de acceso público relacionada con la actual enfermedad de COVID-19, para facilitar la investigación en la pandemia. Según el anuncio [44] de AWS, “este lago de datos permite realizar rápidamente análisis de datos en el lugar de perder tiempo extrayendo y discutiendo datos de todas las fuentes de datos disponibles.”

Los datos están recopilados y se actualizan regularmente de The New York Times, Universidad de John Hopkins, una librería vasta de artículos de investigación relacionado con la enfermedad del Instituto de Allen para IA y mucho más. El lago de datos está publicado en Amazon S3 bucket, totalmente abiertos para cualquier análisis por una cuenta de AWS. Según la fuente se puede realizar “análisis de tendencias, búsquedas de palabras clave, análisis de preguntas y respuestas, crear modelos de Machine Learning o realizar análisis personalizados” utilizando las herramientas de AWS como: Amazon Athena y Amazon Redshift Spectrum como BDD y recuperación de información, AWS Glue para catalogación de definiciones de esos datos y finalmente Amazon QuickSight para la visualización de datos, tal como podemos ver en la Figura 21 [44].

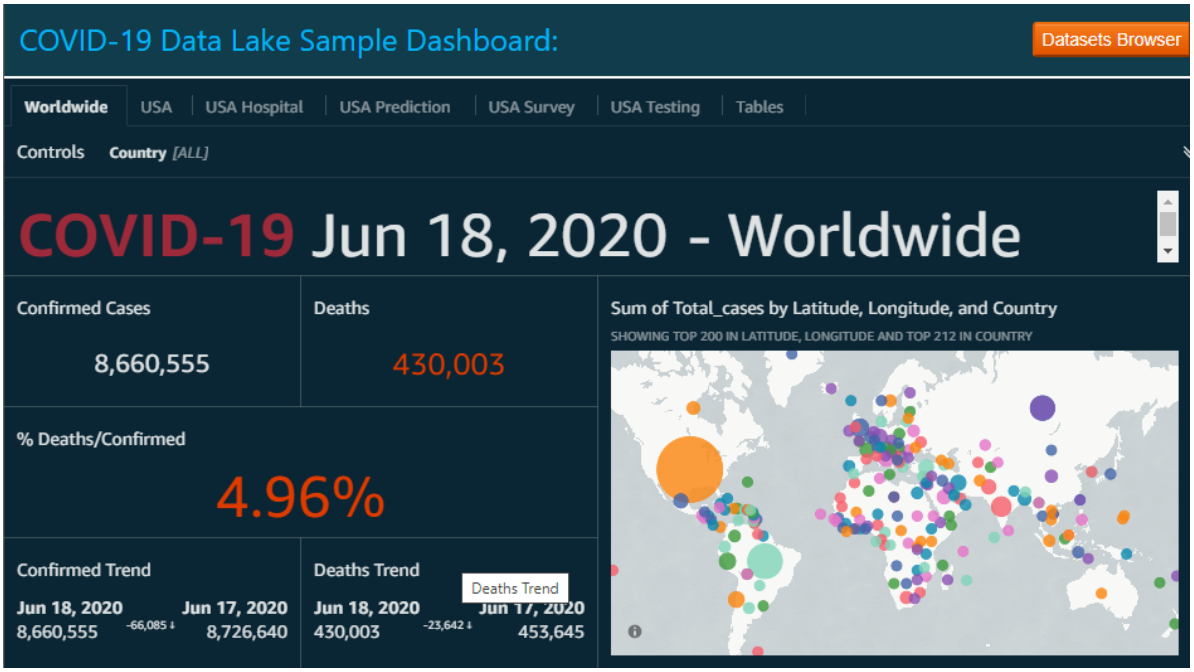


*Figura 21. Visualización de COVID-19 con QuickSight, datos de: seguimiento de casos, tests y camas de hospital. Fuente: [AWS COVID19-lake](AWS COVID19-lake).*

Al contrario de la arquitectura Serverless, la arquitectura tradicional, hubiera sido construida con servicios no gestionados. Aproximadamente tendría una estructura como se muestra en la Figura 22, con los siguientes servicios como instancias de EC2, EBS para los backups, incluido sus instantáneas de estado actual en un bucket de S3. También sería una opción de bases de datos relacionales Redshift SQL o RDS y finalizando con un cortafuego de grupos de seguridad mencionados anteriormente (grupos de seguridad de EC2), que podría cumplir los protocolos de seguridad.

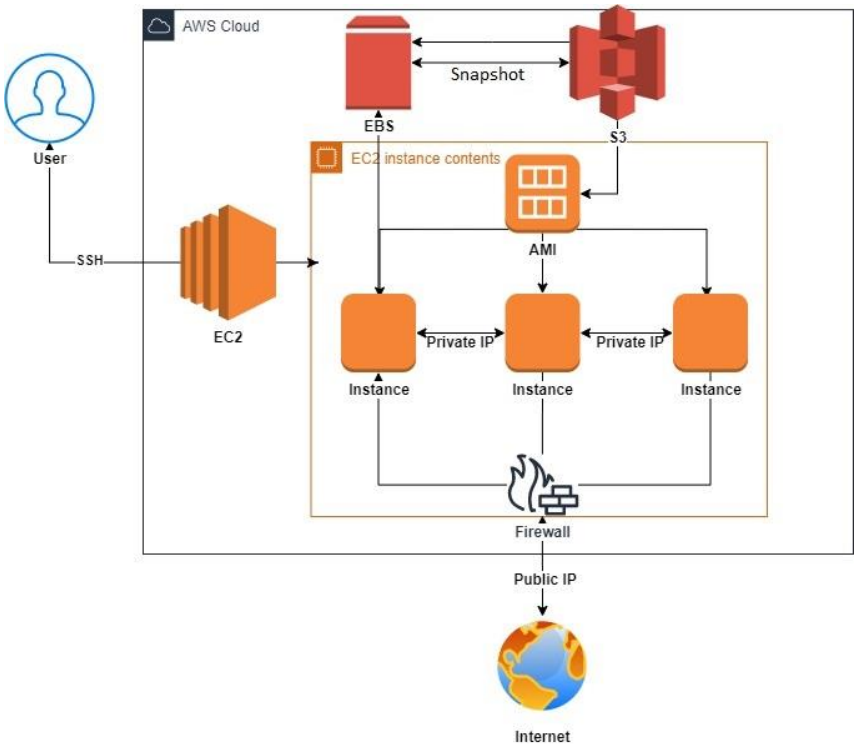


*Figura 22. Arquitectura tradicional para procesamiento de datos masivos.*

### *Internet of Things (IoT) Backends*

AWS IoT es una plataforma que se usa mayormente para conectar dispositivos físicos y hacer que interactúen con otros dispositivos o aplicaciones.

Frost & Sullivan [45] ha nombrado AWS como líder en las plataformas IoT y se considera como un pionero en la industria. Tiene varios servicios para diferentes soluciones: IoT para Edge (IoT para los dispositivos periféricos), IoT Industrial, IoT hogar conectado, AWS IoT y Amazon Alexa.

Ofrece siguientes servicios [46]:

- ***Software del dispositivo:*** *Amazon FreeRTOS, AWS IoT Greengrass.* Se usa para conectar los dispositivos de manera segura, almacenar datos en caché y realizar acciones locales.
- ***Servicios de control:*** *AWS IoT Core, AWS IoT Device Management, AWS IoT Things Graph, AWS IoT Device Defender.* Permite controlar y administrar una gran cantidad de dispositivos diferentes, y que se interactúen de manera segura con otros dispositivos y las aplicaciones en la nube.
- ***Para análisis:*** *AWS IoT Analytics, AWS IoT Events, AWS IoT SiteWise.* Permiten extraer valor de los datos de IoT ejecutando análisis sofisticados en volúmenes masivos de datos.

Veremos un ejemplo [47] recopilado con servicios IoT Core, Lambda y Kinesis de AWS para una solución de los picos de tráfico en diciembre de iRobot (las aspiradoras robóticas Roomba) ejecutando su aplicación orientada al cliente y el backend IoT de sus robots en una arquitectura Serverless de AWS.

En la Figura 23 se muestra claramente el flujo del trabajo de la infraestructura creada, donde:

- Al encender la aspiradora el dispositivo se conecta al AWS IoT Core.
- Utilizan Api Gateway para publicar y asegurar APIs.
- El backend de iRobot usa AWS Lambda para ejecutar código en respuesta a eventos.
- Almacena los datos de gran escala en DynamoDB.
- Para compartir su estado actual del dispositivo (estado de limpieza y la programación) utiliza AWS IoT device shadow.
- Entonces la aplicación se conecta y Amazon Kinesis ingiere flujos de datos en tiempo real del robot.

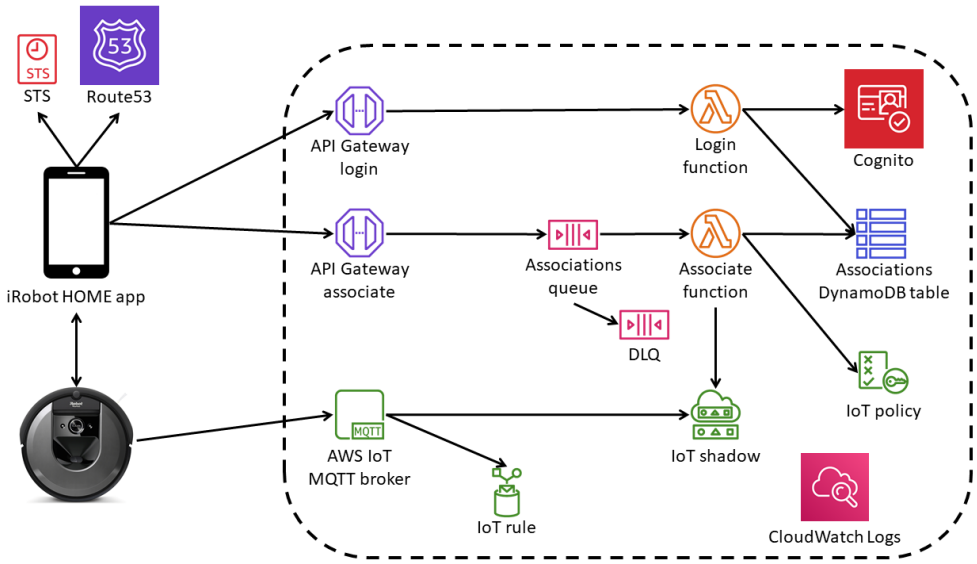


*Figura 23. Solución IoT con arquitectura Serverless para los picos de tráfico de aspiradoras iRobot. Fuente AWS.*

Según AWS [47], gracias a la escalabilidad de la arquitectura Serverless el aumento del tráfico fue controlado y libre de caídas, sin que la empresa planee la escala y el rendimiento de la infraestructura.

Con una arquitectura tradicional para este ejemplo, habría que desarrollar un backend con la siguiente lista de servicios de AWS:

- Se mantendría API Gateway para garantizar la conexión fiable a los usuarios,

- Servicio ELB (Elastic Load Balancer), un balanceador de carga para aguantar los enormes picos de tráfico,
- EC2 con Auto Scaling Group: que permitiría escalar automáticamente las capacidades de las instancias,
- RDS para bases de datos MySQL.

Podemos decir que, en este caso, en las épocas de picos de trafico la empresa hubiera sido gestionado ella misma en momentos de alta carga del sistema y con poca tolerancia a fallos.

Resumiendo, la lista de uso de la arquitectura Serverless es muy larga. Se puede implementar en muchos casos, como autenticación, procesamiento de datos en tiempo real, DevOps, procesamiento de media, seguridad, Cron Jobs, Machine Learning, Chatbots, notificaciones en tiempo real, loging, ubicación, etc. Solo usando los servicios de AWS como; Amazon Athena [48], EMR [49], Glue [50], DynamoDB [33], Kinesis [51], QuickSight [52] y Redshift [53]. Sin embargo, los otros proveedores que hemos mencionado anteriormente se vuelven cada vez más competitivos con AWS. En la sección 3.2, veremos los modelos de coste de las arquitecturas mencionadas anteriormente para ver sus costes y la rentabilidad de cada uno.

## 3.2. Modelos de Coste para Arquitecturas de Referencia

Para el análisis de costes hemos elegido en la calculadora AWS [8] la región U.S. East, Virginia del Norte, donde los precios son asequibles en comparación con otros regiones. Como hemos visto en la Tabla 2, la capa gratuita para EC2 solo cubre los primeros 12 meses. Sin embargo, para los servicios Serverless la capa gratuita no tiene caducidad si no se pasa el límite de uso indicado (AWS Lambda, DynamoDB, etc).

*Tabla 4 Precios de EC2 de familia t3 y m5 (Linux), bajo demanda.*

| Instancia | CPU virtual | Memoria (GiB) | Almacenamiento de la instancia (GiB) | Ancho de banda de red (Gbps) | Precio bajo demanda |
|---|---|---|---|---|---|
| t3.micro | 2 | 1 | Solo EBS | Hasta 5 | $0,0104 |
| t3.small | 2 | 2 | Solo EBS | Hasta 5 | $0,0208 |
| t3.medium | 2 | 4 | Solo EBS | Hasta 5 | $0,0416 |
| t3.large | 2 | 8 | Solo EBS | Hasta 5 | $0,0832 |
| m5.large | 2 | 8 | Solo EBS | Hasta 10 | $0,096 |
| m5.xlarge | 4 | 16 | Solo EBS | Hasta 10 | $0,192 |
| m5.2xlarge | 8 | 32 | Solo EBS | Hasta 10 | $0,384 |
| m5.4xlarge | 16 | 64 | Solo EBS | Hasta 10 | $0,768 |

Los tipos de instancias de EC2 que hemos elegido para la comparación entre los casos de uso de arquitectura no-Serverless y Serverless son de familia t3 y m5. En la

Tabla 4 están los precios de los dichos tipos de instancias que han sido utilizado en las comparaciones económicas.

Las instancias de familia t3 se han elegido para las arquitecturas de aplicación web y IoT, ya que a pesar de que son de última generación, según AWS [54], también ofrecen un equilibrio entre recursos informáticos, de memoria y de red. Las instancias tipo t3 están diseñadas para aplicaciones con picos temporales de uso de CPU, lo que para casos de uso como aplicaciones web e IoT es necesario.

Las instancias de familia m5 han sido elegidas para la arquitectura tradicional de Big Data. Es un tipo de instancia dedicada para bases de datos de tamaño medio, de gran ancho de banda de red que podría ser adecuado para este tipo de caso de uso.

### *Aplicación Web*

*Coste de Arquitectura Serverless*

El caso de uso que hemos visto en la sección 3.1 es un simple ejemplo de cómo podría ser una aplicación web. Durante la valoración de su coste hemos optado a un escenario donde la arquitectura no pasaría los límites de la capa gratuita de los servicios Serverless.

Para la arquitectura Serverless el coste de la función Lambda fue calculada sin aplicar la capa gratuita, ya que elegimos un escenario donde se pasa el límite de la capa gratuita (1 millón de peticiones) hasta 2,2 millones de peticiones, con la memoria de 128 MB alojada y 1000 ms de duración. Desde ese momento AWS empieza a cobrar por cada ejecución de una función (0,20 por cada millón de peticiones) con la concurrencia incluida (precio de una concurrencia - 0,0000041667$ por cada GB-segundo). Tal como se ve en la Tabla 5, el coste de semejante arquitectura con las siguientes características costaría 12,63$ al mes. Ya que la mayoría del coste de la arquitectura procedería de la función Lambda y el alojamiento de la página web en Amazon Amplify. El resto de los servicios permanecerían con la capa gratuita, por no haber sobrepasado los límites.

*Tabla 5. Coste de la arquitectura Serverless para Aplicación Web.*

| Servicios | Características | Precios/mes |
|---|---|---|
| AWS Lambda | 1 función (más de 2 millones de peticiones) con 1 concurrencia (Sin capa gratuita) | $0,26 |
| Amazon Cognito | 50 000 UMA* (sin superar la capa gratuita) | $0,00 |
| API Gateway | 1 millón de llamadas a API (sin superar la capa gratuita) | $0,00 |
| Amazon Amplify | 5GB de almacenamiento, 15 GB de uso por mes y 1000 min. de creación/mes (1 año de capa gratuita) | $12,37 |
| DynamoDB | 25 GB de almacenamiento, 25 unidades de capacidad de lectura y escritura. (sin superar la capa gratuita) | $0,00 |
| **Coste total** | | **$12,63** |

**Usuarios Mensuales Activos*

*Coste de Arquitectura no-Serverless*

Para la arquitectura de la *aplicación web* elegimos dos instancias de t3.small, con las siguientes características: 2 GiB de memoria, 2 CPUs virtuales, EBS (Elastic Block Storage) de 5 GB, que permitiría tener tolerancia a fallos.

En la Tabla 6 se ve una gran diferencia de coste total comparado con la Tabla 5 que es una arquitectura Serverless. A pesar de que elegimos instancias de tamaño no muy grande, el coste se ha aumentado por los servicios añadidos como los balanceadores de carga (ELB) y servicio de DNS (Route 53).

*Tabla 6. Coste de la arquitectura no-Serverless para Aplicación Web.*

| Servicios | Características | Precios/mes |
|---|---|---|
| EC2 | 2 instancias de t3.small (Linux) h/mes | $15,18 |
| Elastic Block Storage (EBS) | 1 volumen de 5GB/mes | $3,00 |
| Elastic Load Balancer (ELB) | 2 balanceadores de carga (tipo clásico) h/mes | $48,18 |
| Auto Scaling | Auto escalado sin cargos. | $0,00 |
| Route 53 | con 1 flujo de tráfico/mes | $52,20 |
| ElastiCache | tipo de instancia t3.micro h/mes | $24,82 |
| RDS MySQL | 1 instancia de db.t3.micro h/mes | $25,02 |
| **Coste total** | | **$168,40** |

***IoT***

*Coste de Arquitectura Serverless*

Dado que la empresa iRobot es una empresa multinacional y la cantidad de los robots adquiridos aumenta durante la época de grandes ventas, la cantidad de los dispositivos conectados se dispara. Por lo tanto, hemos aproximado unos 10 millones de peticiones de conexiones en la arquitectura Serverless de IoT (ver la descripción en la pág. 46).
Teniendo dos funciones Lambda (con las mismas características que la función de la aplicación web) con las concurrencias y superando el límite de la capa gratuita, el coste de la ejecución mensual de las funciones sería 16,02$ (Tabla 7). El mayor peso del coste total llevaría el servicio completo del IoT por 198,63$ al mes.
En esta arquitectura fueron utilizados siguientes características del IoT Core que influyeron al coste del servicio y al coste total de la arquitectura Serverless, que son Figura 23:

- *IoT Message Broker* – un intermediario de mensajes de alto rendimiento con un protocolo de comunicación MQTT (Message Queuing Telemetry Transport).

- *IoT Rule* – permite a los dispositivos interactuar con los servicios de AWS como Lambda, CloudWatch, DynamoDB, etc.
- *IoT Shadow* – actualización de estado de los dispositivos.
- *IoT Policy* (autorización) – control del acceso al plano de datos de AWS IoT Core.

El precio por cada característica es:
- *Servicio de conectividad por 1 dispositivo* - 0,08$ (por 1 000 000 minutos de conexión),
- Servicio de mensajería – 1,00$ (hasta 1000 millón de mensajes). En nuestro caso elegimos 1000000,
- *IoT Rule* – 0,30$ por millón de reglas o acciones ejecutadas (reglas activas 0,15$ + acciones ejecutadas 0,15$)
- *IoT Shadow* – 1,25$ por millón de operaciones

Los cálculos para 1 dispositivo conectado a AWS IoT Core durante 30 días son [55]:

Minutos de conexión = 1 conexión * 60 minutos/hora * 24 horas/día * 30 días = 43200 minutos de conexión
Cargos totales de conectividad = 43200 minutos de conexión * 0,08$ /1.000.000 minutos de conexión = 0,003456$

Como hemos comentado anteriormente, en los periodos de navidades la cantidad de los dispositivos conectados se disparan, sin embargo no sabemos la cantidad exacta de conexiones (deducimos que son muy grandes). Por tanto elegimos siguiente escenario como ejemplo, donde tendríamos 50.000 dispositivos conectados a AWS IoT Core durante 1 mes, que nos costaría 180$ redondeado con las características añadidas.

*Tabla 7. Coste de la arquitectura Serverless IoT.*

| Servicios | Características | Precios/mes |
|---|---|---|
| IoT Core | Con todos sus cargos | $180,00 |
| Cognito | Más de 10 millones UMA | $0,0025 |
| API Gateway | Más de 300 millones de peticiones/mes | $0,90 |
| Lambda | 2 funciones (10 millones de peticiones), 2 concurrencias | $16,20 |
| DynamoDB | Bajo demanda sin la capa gratuita (por 1 millón de escrituras y lecturas) | $1,50 |
| Amazon Kinesis | Primeros 500 TB /mes | $0,029 |
| **Coste total** | | **$198,63** |

*Coste de Arquitectura no-Serverless*

Elegimos una instancia reservada por un año de t3.medium (ya que sale más rentable que una bajo demanda), con 4 GiB de memoria y 2 CPUs virtuales, con pico de tráfico mensual. Se cree que dos instancias con estas características deberían ser suficiente

para una arquitectura que tiene pico de carga en periodo de navidades y después de salida de un nuevo modelo de su producto.

En la Tabla 8 podemos ver como habrían subido los costes de la arquitectura *IoT* si la empresa mantuviera la arquitectura tradicional. También la gestión de los servidores sería evidente, ya que el riesgo de los casos de caídas de servidores por la carga de tráfico seguiría existiendo.

*Tabla 8. Coste de la arquitectura no-Serverless para IoT.*

| Servicios | Características | Precios/mes |
|---|---|---|
| EC2 | 2 instancias de t3.medium (Linux) h/mes (reservado) | $114,32 |
| EBS | 1 volumen de 5GB/mes | $9,00 |
| API Gateway | Más de 300 millones de peticiones/mes | $0,90 |
| Route 53 | Con más de un flujo de datos (p.ej. 10) | $502,20 |
| Auto Scaling | Auto escalado sin cargos. | $0,00 |
| Redshift SQL | Instancia dc2.large, memoria 15GiB (reservado), 2TB de backup | $193,10 |
| **Coste total** | | **$819,52** |

### *Big Data*

*Coste de Arquitectura Serverless*

Para las arquitecturas de Big Data hemos intentado aproximadamente estimar el coste de una arquitectura de Big Data cerca a la realidad, ya que con los datos que tiene el caso de COVID-19 [44] es un poco difícil de calcular teniendo poca información sobre los volúmenes de datos y la cantidad de consultas que podría recibir al día. Se ha calculado cómo sería y cuánto costaría una arquitectura Serverless de Big Data. Como podemos ver en la Tabla 9 la parte más cara de una arquitectura Serverless es el almacenamiento de S3 y la base de datos de Redshift Spectrum. Ya que elegimos con una capacidad bastante grande para el alojamiento de los datos.

*Tabla 9. Coste de la arquitectura Serverless para Big Data.*

| Servicios | Características | Precios/mes |
|---|---|---|
| Amazon Athena | 1000 consultas/semana, datos escaneados 100MB | $2,07 |
| Amazon Redshift Spectrum | Instancia dc2.large, memoria 15GiB (bajo demanda), 2TB de data escaneado | $192,50 |
| S3 | Almacenamiento estándar 10TB | $235,52 |
| AWS Glue | catálogo de datos 2 millones/mes | $43,75 |
| Amazon QuickSight | Máximo lector/mes para el uso ilimitado | $5,00 |
| **Coste total** | | **$478,84** |

*Coste de Arquitectura no-Serverless*

En este caso es más apropiado usar una instancia que pudiera tener gran capacidad de cómputo como m5.large, ya que es adecuado para procesamiento de datos con memoria adicional y almacenamiento de cache. Tiene 8 GiB de memoria, dos CPUs virtuales, hasta 10 Gigabits de ancho de banda de la red. Estas características permitirían mantener la baja latencia de las consultas de bases de datos y lo más importante procesamiento de datos ilimitados de tiempo (Tabla 10).

*Tabla 10. Coste de la arquitectura no-Serverless para Big Data*

| Servicios | Características | Precios/mes |
|---|---|---|
| EC2 + Snap shots | 2 instancias de m5.large (Linux) h/mes (reservado) | $87,60 |
| Elastic Block Storage (EBS) | 1 volumen de 10GB/mes | $4,50 |
| Elastic Load Balancer (ELB) | 2 balanceadores de carga | $48,18 |
| Redshift SQL | Instancia dc2.large, memoria 15GiB (reservado), 2TB de backup | $193,10 |
| Auto Scaling | Auto escalado sin cargos. | $0,00 |
| **Coste total** | | **$333,38** |

Es claro que el mayor peso del coste total llevaría las instancias de m5.large y el base de datos SQL, lo importante es que no sobrepasa al coste de la arquitectura Serverless.

Aunque la diferencia entre los precios de las arquitecturas Big Data (tradicional y Serverless) no es muy grande (que es $145,46), la arquitectura tradicional se ve más rentable que Serverless.

Como hemos visto, la arquitectura Serverless en muchos casos de uso puede ser bastante rentable. Sin embargo, para casos de uso como Big Data, donde hay grandes cargas de trabajo y largos tiempos de procesamiento, con el modelo Serverless la arquitectura podría ser económicamente poco rentable. En este caso se recomienda utilizar una arquitectura tradicional con instancias EC2.

## 3.3. Caso práctico

Esta sección está dedicada al caso práctico de propia elaboración, en el que fue desarrollada una aplicación para procesamiento y visualización de datos en tiempo real de la ciudad de Madrid, que visualiza el nivel de Dióxido de Carbono.

Para realizar este proyecto se utilizaron los datos de la calidad del aire de Madrid, que generan los sensores instalados por toda la ciudad, a través del Portal de Datos Abiertos de Madrid [46].

Los requisitos técnicos del proyecto son:

- Capturar los datos desde el portal en formato archivos .csv a través de URL pública,
- Almacenar los archivos en un repositorio, al menos 1 GB de capacidad,
- Ejecución de tareas de forma periódica, cada 1 hora,
    - Captura y almacenamiento de datos mediante un script de Python
    - Procesamiento de los datos mediante un script de Python
- Alojamiento de una página web estática para publicación de la visualización de los datos, con acceso público.

Para satisfacer los requisitos del proyecto con una arquitectura tradicional que es despliegue de una instancia EC2 sería necesario:

- Conocimiento de administración de sistemas Linux y programación Bash para las tareas mantenimiento del sistema y de Cron respectivamente.
- Instalación y mantenimiento del entorno de ejecución Python.
- El servidor debería estar activo 24/7, lo que nos llevaría a un derroche de gasto por inactividad ya que el 99% del tiempo estaría latente sin realizar ningún trabajo.

En cuanto a la arquitectura Serverless, para completar los requisitos solo se requería:

- 2 buckets de almacenamiento de S3.
- Funciones Lambda que ejecutarían los dos scripts de código Python.
- Métricas de CloudWatch para la ejecución periódica de las tareas.
- El modelo Serverless abstrae la instalación y mantenimiento de los sistemas subyacentes como S.O., entornos de ejecución, Cron, etc.
- Solo pagamos por el uso, por lo que evitamos el coste de los periodos de inactividad que en caso de nuestra aplicación son largos.

Considerando todos los factores y requisitos del proyecto concluimos que la arquitectura Serverless la mejor solución.

Hay que tener en cuenta también que el proyecto no es un análisis comparativo de la calidad de aire, sino un ejemplo de que cómo se podría crear una aplicación de análisis y visualización de datos utilizando los servicios Serverless. Para un análisis más preciso habría que ver las condiciones de cómo se han tomado los datos, considerando unos factores importantes como:

- Tipos de aparatos que han sido utilizados,
- Numero de sensores utilizados,
- La ubicación de los sensores, etc.

Y como es una demo de supuesto análisis, los factores dichos anteriormente no los consideraremos en este proyecto.

La idea del proyecto es coger los datos de la calidad de aire, precisamente el nivel de Dióxido de Nitrógeno ($NO_2$), generado por los sensores ubicados en Madrid,

procesarlos y visualizarlos en un gráfico multilínea que iba ser publicado en una página web estática en tiempo real (Figura 24).

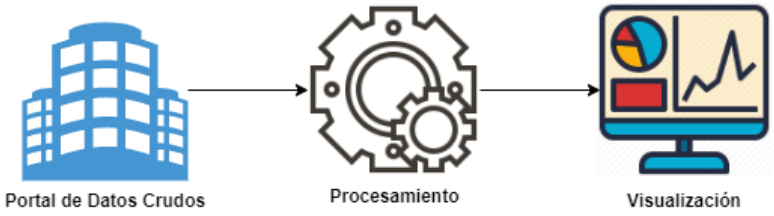


*Figura 24. La visión del proyecto. Propia elaboración.*

*Primero veremos el dataset con que vamos a tratar para este proyecto.* El portal [56] ofrece tres formatos de descarga del conjunto de datos (.txt, .csv, .xml), por lo tanto, elegimos el formato .csv, ya que es más fácil de tratar para la visualización (Figura 25).

28,079,004,01,38,02,2019,01,01,00023,V,00045,V,00028,V,00037,V,…
Los caracteres se corresponden con lo siguiente:

| PROVINCIA | MUNICIPIO | ESTACIÓN | MAGNITUD | TÉCNICA | PERIODO ANÁLISIS | AÑO | MES | DÍA | DATO | CÓDIGO DE VALIDACIÓN |
|---|---|---|---|---|---|---|---|---|---|---|
| 28 | 079 | 004 | 01 | 38 | 02 | 2019 | 01 | 01 | 00023 | V |

*Figura 25. La estructura de .csv de los datos descargados. Fuente: Intérprete de ficheros de datos horarios.*

El conjunto de datos tiene múltiples variables. Dado que no vamos a necesitar todos, crearemos otro conjunto de datos donde estarán las variables que necesitamos, como: código de la estación, código de la magnitud de Dióxido de Nitrógeno y los resultados horarios. Para que estén preparados de ser utilizados en la visualización. Este proceso lo veremos de cerca más adelante.

Como podemos ver en *el diagrama* (Figura 26), para este proyecto seguimos un modelo de arquitectura de ETL (extraer, transformar y cargar), un sistema que captura, organiza y enruta datos para que puedan usarse con el objetivo de obtener información. Sin embargo, nuestro proyecto es una versión que no llega a niveles de Big Data, ya que nuestros datasets tienen peso bastante reducido.

Primero los datos comienzan siendo ingeridos a S3 y luego transformados con las funciones Lambda y luego, en una página web estática situada en otro bucket de S3 los mismos datos serán visualizados usando librería D3.js [57]. Y todo esto se ejecuta automáticamente cada 1 hora con Amazon CloudWatch Events.

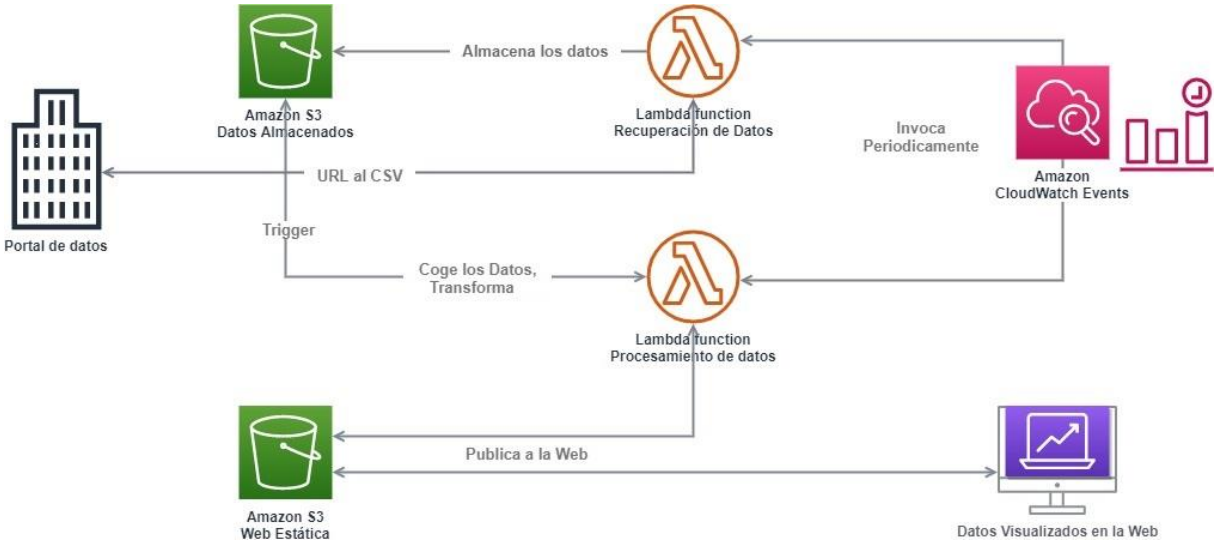


*Figura 26. El diagrama del proyecto visualización de datos, con arquitectura Serverless. Propia elaboración con draw.io.*

Ahora, veremos paso a paso la creación de la misma aplicación.

Nuestra arquitectura se divide en 3 partes: captura y almacenamiento de datos, procesamiento de datos y visualización de datos.

***Almacenamiento y procesamiento de datos.***

Esta parte del proyecto es el más importante, ya que se basa en dos funciones de Lambda que desencadena los eventos para la descarga y procesamiento de los datos obtenidos.

Echamos un vistazo a la función desde dentro.

Una función Lambda consiste en 3 importantes aspectos (Figura 27):

- ***handler***. Que es un controlador que genera la función.
- ***Event***: es información detallada sobre el evento durante la ejecución.
- ***Context***: contiene métodos y propiedades disponibles para interactuar con información de entorno de ejecución (como el límite de la memoria, grupo de registro, etc.).

El código para nuestras funciones ha sido escrito en lenguaje Python 3.6 con dependencias de Boto3 [58], Requests [59] y Pandas [60] (para la edición del .csv).

Antes de subir una función Lambda se crea un paquete de implementación (un fichero .zip con dependencias y el código desarrollado). Después de crear los paquetes de implementación se envía al servicio Lambda con las herramientas de línea de comandos AWS CLI o a través de AWS Management Console por navegador.

lambda-get-data. ×

```python
from io import BytesIO
import os
import requests
import boto3

def handler(event, context):

    url = "http://www.mambiente.madrid.es/opendata/horario.csv"
    response = requests.get(url)
    data = response.content
    print(data)

    s3 = boto3.client('s3')
    s3.put_object(Bucket='slsbucket393', Key='horario.csv', Body=data)

```

8:12 Python Spaces: 4

*Figura 27. Función Lambda para la descarga del fichero .csv desde el portal.*

Como hemos dicho anteriormente, ***la primera función*** con las librerías de boto 3 y requests se dispara con un trigger que descarga cada una hora el fichero .csv en un bucket de s3 que se llama “slsbucket393” (Figura 27).

***La segunda función***: coge el fichero .csv desde el slsbucket393, con la librería panda edita el dataset eliminando las columnas innecesarias, quedándonos con solo los datos de dióxido de nitrógeno, con los resultados de las 24 estaciones y juntando las columnas de la fecha y hora en una columna. Finalmente se crea un nuevo .csv que contiene: la fecha/hora de la medición y las estaciones con sus resultados (Figura 28). Que se guarda en otro bucket de S3 con nombre “staticwebbucket939” donde se sitúa la página web estática, para estar preparado de ser visualizado.

| | 3 | 15 | 22 | 27 | 33 | 43 | 55 | 61 | 67 | 72 | ... | 86 | 90 | 95 | 100 | 104 | 108 | 119 | 127 | 131 | 135 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2020-2-12-H01 | 45.0 | 60.0 | 55.0 | 47.0 | 47.0 | 44.0 | 63.0 | 48.0 | 61.0 | 48.0 | ... | 56.0 | 46.0 | 53.0 | 41.0 | 55.0 | 55.0 | 50.0 | 52.0 | 44.0 | 59.0 |
| 2020-2-12-H02 | 44.0 | 64.0 | 55.0 | 48.0 | 49.0 | 41.0 | 55.0 | 48.0 | 63.0 | 46.0 | ... | 53.0 | 46.0 | 52.0 | 41.0 | 53.0 | 52.0 | 48.0 | 51.0 | 43.0 | 57.0 |
| 2020-2-12-H03 | 44.0 | 65.0 | 58.0 | 49.0 | 47.0 | 39.0 | 44.0 | 48.0 | 58.0 | 44.0 | ... | 51.0 | 48.0 | 46.0 | 44.0 | 53.0 | 48.0 | 50.0 | 55.0 | 37.0 | 58.0 |

*Figura 28. La tabla editada con resultados horarios de NO2.*

El servicio AWS CloudWatch evento juega una parte importante en este proyecto, ya que todo esto se mantiene en ejecución gracias a los triggers que definimos para cada función. Los triggers tienen sus roles asignados para cada función (con permisos de acceso a los servicios de AWS). Creamos reglas para invocar objetivos (Targets), en nuestro caso funciones, basados en eventos. Cada evento de CloudWatch tiene configurado para disparar los Targets cada 60 minutos (Figura 29).

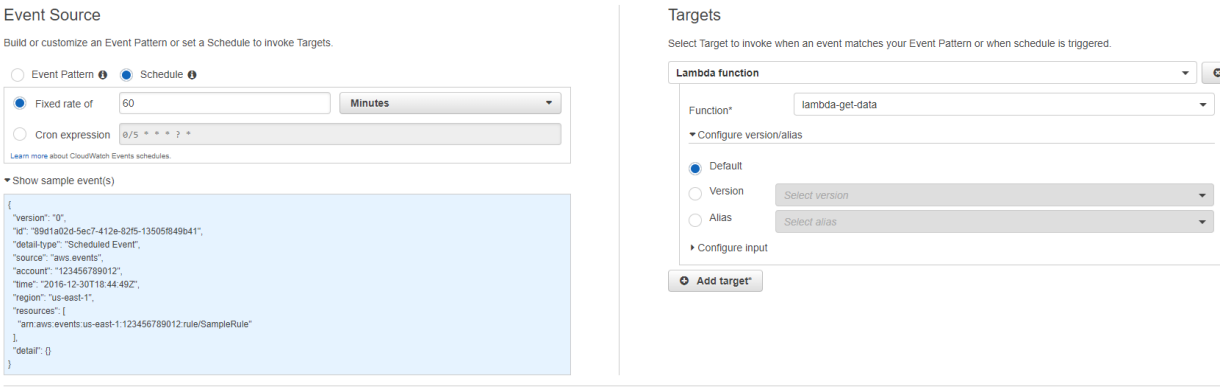


*Figura 29. Configuracion de reglas de CloudWatch Events.*

### *Visualización y publicación de datos:*

Con la librería de D3.js creamos el grafico multilínea (Figura 31) y el código JavaScript del grafico añadimos al fichero .html principal de la página web. Para la página web fue descargada previamente una plantilla desde Bootstrap [61] y editada para la publicación.

La página web ha sido alojada en un bucket de S3 asignado solo para este propósito, cambiando los propiedades y permisos del bucket, haciéndolo con acceso público.


*Figura 30. La portada de la página web donde está publicada el gráfico. Fuente: [Eco Datas Madrid](#).*

Finalmente entrando con el enlace del S3 bucket [62] donde aloja la página web podemos ver la descripción de los datos y el grafico de los mismos en tiempo real (Figura 30).

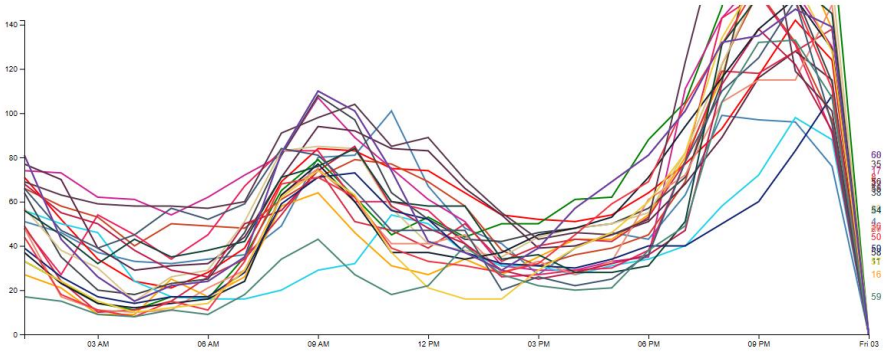


*Figura 31. Gráfico del nivel de $NO_2$ de Madrid el día 21 de febrero 2020.*

Un dato curioso que ha ocurrido con el medioambiente de Madrid durante la investigación. Desde las mitades de 2019 (septiembre) y los principios de 2020 hubo días en los que el aire de Madrid estaba bastante contaminado. Sin embargo, al empezar el confinamiento por la pandemia los niveles de $NO_2$ bajaron drásticamente. Por ejemplo, podemos ver la comparación de los gráficos de dos fechas diferentes, entre 21/2/20 y 5/5/20. Donde el día 21 de febrero, en algunos puntos de la ciudad, el nivel de $NO_2$ llegó hasta 230.0 microgramos/metro cúbico ($\mu g/m^3$) (Figura 31). Mientras tanto, la contaminación por $NO_2$ no pasó alrededor de 57.0 $\mu g/m^3$ (Figura 32).

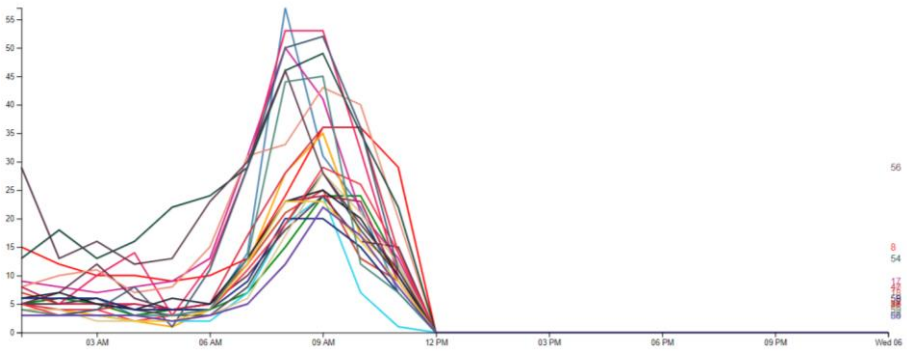


*Figura 32. Gráfico del nivel de $NO_2$ de Madrid el día 5 de mayo 2020.*

Finalmente, como hemos visto anteriormente, utilizando solo dos servicios FaaS de AWS hemos podido desarrollar una aplicación web de análisis de datos tan solo desarrollando dos tipos de código, sin meternos en la gestión de la infraestructura,

tomar decisiones de cuantas estancias necesitaríamos para ejecutar nuestros códigos, para tener alta disponibilidad o preocuparnos de que se nos pasemos del presupuesto si ejecutásemos una estancia tradicional a 24h/7.

### 3.3.1. Costes

Después de crear una arquitectura Serverless, utilizando la plataforma de AWS, analizamos la parte económica de la arquitectura: comparando los costes de uso de AWS Lambda con un despliegue de una instancia de EC2.

Utilizando la calculadora de costes Serverless creada por Peter Sbarski [63], comparamos el servicio Serverless de los proveedores mencionados anteriormente, más añadido el OpenWhisk del IBM, para saber el coste diferencial de una ejecución de una función en los entornos de otros proveedores.

Definiendo las características de la función ejecutado, podemos ver los precios mensuales para cada proveedor sin la capa gratuita (Tabla 11).

- Número de ejecuciones – 1.000.000
- Tiempo Estimado de la ejecución – 1000 ms
- Tamaño de la memoria – 128 MB
- Peticiones HTTP – yes

Como podemos ver, el coste del AWS Lambda es la más cara comparado con el resto de los servicios. La diferencia que sobresale es el coste por petición HTTP.

*Tabla 11. Comparación precios del ejecución de una función.*

| Proveedor | Coste petición HTTP | Coste Computación | Coste Total |
|---|---|---|---|
| AWS Lambda | $3.70 | $2.08 | $5.78 |
| Azure Functions | $0.20 | $2.00 | $2.20 |
| Google Cloud Functions | $0.40 | $2.31 | $2.71 |
| IBM OpenWhisk | $0.00 | $2.13 | $2.13 |

Como no sabemos la manera del cálculo que hace esta calculadora, por lo tanto, para nuestro proyecto utilizaremos la calculadora oficial del AWS a pesar de nuestras facturas recibidas, que permite ver todos los cálculos en forma detallada, que se puede ver en Anexo 1 [8].

### *Arquitectura Serverless vs EC2*

Es importante que el desarrollador entienda las implicaciones de costes antes de elegir una tecnología en particular. El coste de ejecutar una aplicación depende de muchos factores, como el alojamiento, la base de datos, etc. También hay que tener en cuenta que el tiempo ejecutado de una función Lambda depende del tipo de lenguaje (runtime) y de las dependencias de terceros.

Para comparar los costes entre arquitectura Serverless y una instancia de EC2, veremos los cálculos hechos para cada uno.

*Arquitectura Serverless*

Nuestra arquitectura se considera como un caso de bajo uso de cómputo, ya que tenemos:

- procesamiento de datos (Lambda),
- alojamiento (S3).
- trabajos CRON programados (CloudWatch events),

Dado que la mayor parte del trabajo de cómputo se centra en dos scripts de código que ejecutan nuestras funciones Lambda, aquí es donde se esperaba mayor coste. Sin embargo, no lo es.

Nuestras dos funciones Lambda se ejecutan 24 veces al día (cada 1 hora). Por lo tanto, en un mes se ejecutan 1460 veces (730 x 2). Esto ni si quiera llega a las 1500 ejecuciones al mes. Tampoco hemos necesitado añadir Provisioned Concurrency, a pesar de que nuestras funciones tienen largos periodos de inactividad (1 hora), pues el retardo inicial de la primera invocación (cold start) en el que se incurre es despreciable para el caso que nos ocupa.

Amazon S3 almacena nuestro contenido estático como HTML, CSS y JavaScript, y los datasets del portal. El coste de S3 para el almacenamiento estándar de 1 GB (lo que hemos elegido, ya que el contenido de nuestros buckets no sobrepasa de 1 GB) es de $ 0,02.

Aparte de ejecutar los eventos, los costes para CloudWatch dependen también de los logs. Estos logs se crean a través de CloudWatch Logs y se almacenan. Se puede controlar la cantidad de los logs para optimizar el coste.

Basándose en las características indicadas anteriormente para cada servicio, tenemos los siguientes costes (Tabla 12).

*Tabla 12. Los costes de los servicios usado para el proyecto (por mes).*

| Servicio de AWS | Caracteristicas definidas | Coste total al mes |
|---|---|---|
| **AWS Lambda** | **2 funciones**<br>Memoria asignada – 128 MB<br>No. de invocaciones (x 2) – 1460 veces/mes<br>Duración de ejecución – 1000 ms | **Sin superar la capa gratuita $0.00** |
| **Amazon S3** | **2 buckets de S3**<br>Almacenaje estándar: 1 GB por mes.<br>PUT, POST, requests: 24 | **S3 Standard cost (monthly): $0,04** |
| **CloudWatch** | **CloudWatch events:** 2 métricas | **Sin superar la capa gratuita $0,00** |
| **Coste Total** | | **$0,04** |

*EC2*

Para una arquitectura tradicional si hubiéramos elegido una instancia a demanda más pequeña de t2.nano para alojar nuestros datasets y la página web, el coste mensual sería de 4,23$. *(1 instancia x 0,0058 $ x 730 horas en un mes = 4,23$ (bajo demanda).*

Y por un Amazon Elastic Block Storage (EBS) de 5 GB, que nos costaría 0,25$/mes *(5 GB x 0,05 $ x 1 instancias = 0,25 $)*, por muy poca diferencia, el coste total de una instancia de EC2 ***sería 4.48 $.***

Como podemos ver el coste mensual de nuestra aplicación no llega ni a 0,1$ al mes comparado con una instancia de EC2 con la ejecución de 24/7.

Mas importante incluso que los costes derivados del alquiler de recursos informáticos en la nube, es el coste de los recursos humanos necesarios para la implementación y mantenimiento de estos sistemas.

En una arquitectura basada en IaaS los costes de instalación y mantenimiento son altos, ya que tiene que ser realizados por técnicos especializados que tenemos que contratar para realizar estas tareas. Creando además una dependencia hacia al profesional. En una arquitectura Serverless eliminamos todos los costes y dependencias relativas a la instalación y mantenimiento de la arquitectura subyacente.

La arquitectura Serverless puede ser muy beneficiosa en desarrollo de aplicaciones, sin embargo, también tiene sus puntos inconvenientes. Algunas empresas tienen dudas de si merece la pena apostar a ese tipo de arquitectura, ya que los lleva al "vendor lock-in". Además, en los casos que necesitarán altos recursos de cómputo, como trabajos ETL, EC2 podría ser una mejor opción. Por otro lado, es perfecto para

casos de uso de bajo cómputo, como nuestra aplicación, trabajos programados, autenticación, chatbots, etc.

Citando el artículo de BBVA [64] donde se ha realizado un estudio del impacto económico de la arquitectura Serverless se dice, *"Con perfiles de tráfico donde las peticiones llegan en intervalos periódicos, y un número total de peticiones pequeño, la arquitectura 'Serverless' parece ser una buena elección en términos de coste, velocidad de despliegue y esfuerzo. De esta manera, Lambda es probablemente la solución a elegir si la aplicación tiene períodos de inactividad suficientemente largos."*

Durante la investigación hemos llegado en la conclusión que elegir los servicios sabiamente, mapeándolos y calcular el coste de su proyecto es fundamental antes de comenzar con la arquitectura Serverless. Ya que no hay una talla única para todos.

# 4. Conclusión

Podemos decir que la arquitectura Serverless es una etapa evolutiva de Cloud Computing que es nueva, innovadora y poca gente lo conoce muy bien.

Surge una pregunta, ¿Deberíamos concentrarnos solo en construir arquitecturas Serverless?

Al final de la investigación hemos llegado a la conclusión, en que el modelo Serverless no se puede implementar en todos los casos, ya que, antes de nada, hay que considerar los dos factores importantes: técnico y económico.

A nivel técnico, la arquitectura Serverless no siempre puede ser una buena solución. No es adecuado para grandes cargas de trabajo, donde se necesita una gran potencia de procesamiento. Esto se debe a los límites de la memoria y el tiempo de la ejecución de Lambda (512 MB de espacio en disco efímero, 3008 MBs de RAM y 15 min. de duración de ejecución), tomado como ejemplo el servicio de AWS. En este caso sale más efectivo desplegar instancias de EC2. Sin embargo, si la carga de trabajo es ligera y si hay largos períodos de inactividad de la aplicación, una arquitectura Serverless es recomendable.

A nivel económico, después de ver todos los precios y haber comparado los dos tipos de arquitecturas (Serverless y tradicional), se puede decir que el modelo Serverless es más rentable para las arquitecturas como Aplicaciones Web e IoT. Por otro lado, para arquitecturas como Big Data que necesitan una gran potencia de procesamiento y largo periodo de actividad una solución Serverless no sale tan rentable como se creía.

Desde una perspectiva empresarial, es perfecta para los startups o pequeñas empresas con un capital limitado que no pueden permitirse invertir en infraestructura. Sin embargo, las grandes empresas como Netflix [65] teniendo de 30.000 a 50.000 instancias alquiladas en AWS, también apuestan por la arquitectura Serverless para las soluciones como: codificación de archivos multimedia, backups, seguridad y monitorización.

Serverless resume la infraestructura subyacente y libera muchas preocupaciones de gestión que tiene el desarrollador. Es revolucionario, con sus soluciones alternativas en la nube, y on-premises (como Kubeless [66], Fission Functions [67], Apache Openwhisk [68], Knative [69], etc.). La arquitectura Serverless on-premises es otro tema muy interesante para discutir, que reduce el riesgo de la situación tan temerosa por las empresas “Vendor Lock-in” y problemas con las migraciones.

En resumen, el modelo Serverless permite redefinir la arquitectura de los aplicativos abstrayendo total o parcialmente funcionalidades y reduciendo los costes de puesta en marcha y producción. Sin embargo, como cualquier nuevo paradigma llevará un poco de tiempo de evolución para alcanzar su máximo potencial.

# Bibliografía


[1] Peter Mell and T. Grace, “The NIST Definition of Cloud Computing: Recomendations of the National Institute of Standarts and Technology,” *National Institute of Standards and Technoligy U.S. Department of Commerce*, 2011. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[2] “What’s the Difference between Public, Private, Hybrid, and Community Clouds?,” *AbacusNext.com*. [Online]. Available: https://www.abacusnext.com/blog/whats-difference-between-public-private-hybrid-and-community-clouds.

[3] D. W. Raj Bala, Bob Gill, Dennis Smith, “Magic Quadrant for Cloud Infrastructure as a Service, Worldwide,” 2019. [Online]. Available: https://www.gartner.com/doc/reprints?id=1-2G2O5FC&ct=150519.

[4] “Amazon, Microsoft, Google and Alibaba Strengthen their Grip on the Public Cloud Market,” *Synergy Research Group*. [Online]. Available: https://www.srgresearch.com/articles/amazon-microsoft-google-and-alibaba-strengthen-their-grip-public-cloud-market.

[5] D. Smith, L. Leong, and R. Bala, “Gartner Magic Quadrant for Cloud Infrastructure as a Service, Wordwide,” 2018.

[6] “Cloud Comparison: Amazon (AWS) vs Google Cloud (GCP) vs Microsoft (Azure) | CBT Nuggets,” *Youtube*, 2016. [Online]. Available: https://www.youtube.com/watch?v=342KEaxFVjM&feature=youtu.be.

[7] Amazon, “How AWS Pricing Works,” *White Paper*, 2012. [Online]. Available: https://d0.awsstatic.com/whitepapers/aws_pricing_overview.pdf.

[8] “AWS Pricing Calculator.” [Online]. Available: https://calculator.aws/#/.

[9] “Precios de Amazon S3.” [Online]. Available: https://aws.amazon.com/es/s3/pricing/.

[10] “AWS Pricing,” 2019. [Online]. Available: https://aws.amazon.com/pricing/?nc2=h_ql_pr_ln.

[11] C. E. O. at T. L. Aaron Rallo, “New Research From TSO Logic Shows AWS Costs Get Lower Every Year,” 2018. [Online]. Available: https://aws.amazon.com/blogs/apn/new-research-from-tso-logic-shows-aws-costs-get-lower-every-year/.

[12] M. Winters and G. Wints, “CNCF WG-Serverless Whitepaper v1.0,” 2018. [Online]. Available: https://github.com/cncf/wg-serverless/tree/master/whitepapers/serverless-overview.

[13] I. Baldini *et al.*, “Serverless Computing : Current Trends and Open Problems,” 2017, pp. 1–20.

[14] AWS, “AWS Lambda.” [Online]. Available: https://aws.amazon.com/es/lambda/.

[15] “AWS re:Invent 2014 | (MBL202) NEW LAUNCH: Getting Started with AWS Lambda,” *Youtube*. [Online]. Available: https://www.youtube.com/watch?v=UFj27laTWQA.

[16] Google, “Cloud Functions.” [Online]. Available: https://cloud.google.com/functions.

[17] M. Azure, “Azure Functions.” [Online]. Available: https://azure.microsoft.com/es-es/services/functions/.

[18] “AWS Serverless Application Model (AWS SAM) Specification.” [Online]. Available: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-specification.html.

[19] “Zappa - Serverless.” [Online]. Available: https://github.com/Miserlou/Zappa.

[20] “Serverless go microservices for AWS.” [Online]. Available: https://gosparta.io/.

[21] “Serverless Framework.” [Online]. Available: https://www.serverless.com/.

[22] M. R. John Chapin, *What Is Serverless?* O’Reilly Media, Inc., 2017.

[23] P. Johnson, “Serverless: It’s much much more than FaaS,” 2018. [Online]. Available: https://medium.com/@PaulDJohnston/serverless-its-much-much-more-than-faas-a342541b982e.

[24] “Oracle Functions.” [Online]. Available: oracle.com/es/cloud-native/functions/.

[25] “IBM Cloud Functions.” [Online]. Available: https://www.ibm.com/es-es/cloud/functions.

[26] L. Hecht, “ADD IT UP: FAAS ≠ SERVERLESS,” *The New Stack*, 2018. [Online]. Available: https://thenewstack.io/add-it-up-serverless-faas/.

[27] P. Sbarski, *Serverless Architectures on AWS: With examples using AWS Lambda*. Manning Publications, 2017.

[28] “Lambda sample applications.” [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/lambda-samples.html.

[29] S. Purkayastha, “Tutorial: Realtime iOS Heart Rate Monitor and Dashboard,” 2015. [Online]. Available: https://www.pubnub.com/blog/2015-09-30-tutorial-realtime-ios-heart-rate-monitor-dashboard/.

[30] “New – Provisioned Concurrency for Lambda Functions,” 2019. [Online]. Available: https://aws.amazon.com/blogs/aws/new-provisioned-concurrency-for-lambda-functions/#:~:text=You only pay for the,same rate as normal functions.

[31] “AWS Lambda limits.” [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html.

[32] “Amazon API Gateway.” [Online]. Available: https://aws.amazon.com/es/api-gateway/.

[33] “Amazon DynamoDB.” [Online]. Available:

https://aws.amazon.com/dynamodb/.

[34] “AWS CloudFormation.” [Online]. Available: https://aws.amazon.com/es/cloudformation/.

[35] “AWS Lambda features.” [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-features.html.

[36] “AWS Lambda function scaling.” [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/invocation-scaling.html#throttling-behavior.

[37] “AWS re:Invent 2018 | AWS Lambda Layers, the Runtime API, and Nested Applications,” *Youtube*. [Online]. Available: https://www.youtube.com/watch?v=fDv_RKygOXU.

[38] S. M. Kerner, “Top Serverless Vendors,” 2019. [Online]. Available: https://www.datamation.com/cloud-computing/top-serverless-vendors.html#googlecloudfunctions.

[39] Microsoft, “Introducción a Azure Functions,” 2020. [Online]. Available: https://docs.microsoft.com/es-es/azure/azure-functions/functions-overview.

[40] S. M. Abhishek Kumar, *Serverless Integration Design Patterns with Azure*. Packt Publishing, 2019.

[41] “Build a Serverless Web Application.” [Online]. Available: https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/.

[42] “Web Application Hosting in the AWS Cloud Best Practices,” 2012. [Online]. Available: https://www.ncloud24.com/aws/img/file/AWS_Web_Hosting_Best_Practices.pdf.

[43] L. Petrosyan and E. Gallardo, “Escenarios de uso Serverless Computing.” 2018.

[44] A. D. L. Team, “A public data lake for analysis of COVID-19 data,” 2020. [Online]. Available: https://aws.amazon.com/es/blogs/big-data/a-public-data-lake-for-analysis-of-covid-19-data/.

[45] “Enable the internet of (industry-leading) things.” [Online]. Available: https://pages.awscloud.com/Frost-Sullivan-Report.html?trk=ar_card.

[46] “AWS IoT.” [Online]. Available: https://aws.amazon.com/es/iot/?nc=sn&loc=0.

[47] “The Spikiest Time of the Year Is No Problem for iRobot and AWS IoT.” [Online]. Available: https://aws.amazon.com/es/solutions/case-studies/irobot-iot/.

[48] “Amazon Athena.” [Online]. Available: https://aws.amazon.com/es/athena/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc.

[49] “Amazon EMR.” [Online]. Available: https://aws.amazon.com/es/emr/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc.

[50] “AWS Glue.” [Online]. Available: https://aws.amazon.com/es/glue/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc.

[51] “Amazon Kinesis.” .

[52] “Amazon QuickSight.” [Online]. Available: https://aws.amazon.com/es/quicksight/.

[53] “Amazon Redshift.” [Online]. Available: https://aws.amazon.com/es/redshift/?nc2=type_a&whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc.

[54] “Tipos de instancias de Amazon EC2,” *AWS*. [Online]. Available: https://aws.amazon.com/es/ec2/instance-types/.

[55] “AWS IoT Core pricing.” [Online]. Available: https://aws.amazon.com/iot-core/pricing/?nc1=h_ls.

[56] “Calidad del aire. Datos en tiempo real,” *Portal de datos abiertos del Ayuntamiento de Madrid*. [Online]. Available: https://datos.madrid.es/sites/v/index.jsp?vgnextoid=41e01e007c9db410VgnVCM2000000c205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD.

[57] “D3.js Data-Driven Documents.”

[58] “Boto3.” [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html.

[59] “Requests: HTTP for Humans™.” [Online]. Available: https://requests.readthedocs.io/en/master/.

[60] “Pandas.” [Online]. Available: https://pandas.pydata.org/docs/#.

[61] “Bootstrap Templates & Themes.” [Online]. Available: https://startbootstrap.com/themes/.

[62] “Eco Data Madrid,” *Calidad del Aire de Madrid en Tiempo Real Desarrollado Con la Arquitectura Serverless*. [Online]. Available: https://staticwebbucket939.s3.amazonaws.com/index.html.

[63] P. Sbarski, “Serverless Cost Calculator,” 2020. [Online]. Available: http://serverlesscalc.com/.

[64] BBVA_Labs, “La Economía de las Arquitecturas ‘Serverless,’” 2019. [Online]. Available: https://www.bbva.com/es/economia-arquitecturas-serverless/.

[65] “AWS re:Invent 2014 | Netflix Gains New Efficiencies using AWS Lambda.” [Online]. Available: https://www.youtube.com/watch?v=hU25CIRPIJo.

[66] “Kubless-native serverless.” [Online]. Available: https://kubeless.io/docs/.

[67] “Fission: Open source, Kubernetes-native Serverless Framework.” [Online]. Available: https://fission.io/.

[68] “Apache OpenWhisk.” [Online]. Available: https://openwhisk.apache.org/.

[69] “Knative.” [Online]. Available: https://knative.dev/.

# Anexo

*Anexo 1. Los cálculos de los servicios utilizados realizados por AWS Pricing Calculator* [8].

| Services | Calculation |
|---|---|
| **AWS Lambda** | **Unit conversions**<br>Amount of memory allocated: 128 MB x 0,0009765625 GB in a MB = 0,125 GB<br><br>**Pricing calculations**<br><br>RoundUp (1000) = 1000 Duration rounded to nearest 100ms<br>1,460 requests x 1,000 ms x 0,001 ms to sec conversion factor = 1460,00 total compute (seconds)<br>0,125 GB x 1460,00 seconds = 182,50 total compute (GB-s)<br>182.50 GB-s - 400000 free tier GB-s = -399,817.50 GB-s<br>Max (-399817,50 GB-s, 0) = 0,00 total billable GB-s<br>1460 requests - 1000000 free tier requests = -998,540 monthly billable requests<br>Max (-998540 monthly billable requests, 0) = 0,00 total monthly billable requests<br><br>➢ **Lambda costs - With Free Tier (monthly): 0,00 USD** |
| **S3** | **Tiered price for: 1 GB**<br><br>1 GB x 0,0230000000 USD = 0,02 USD<br>Total tier cost = 0,02 USD (S3 Standard storage cost)<br>24 PUT requests for S3 Storage x 0,000005 USD per request = 0,0001 USD (S3 Standard PUT requests cost)<br>0,023 USD + 0,0001 USD = 0,02 USD (Total S3 Standard Storage, data requests, S3 select cost)<br><br>➢ **S3 Standard cost (monthly): 0,02 USD** |
| **Cloudwhtach** | **Tired price for 2 metrics**<br><br>2 metrics x 0.3000000000 USD = 0.60 USD<br><br>**Free tire**<br>$0,00 per alarm metric month - first 10 alarm metrics<br>First 5GB per month of log data ingested.<br>First 5GB-mo per month of logs storage.<br><br>➢ **CloudWatch Metrics cost (with free tire monthly): 0,00 USD** |