Portada

Resumen / Abstract

Índice español

1. Introducción: breve introducción del proyecto
   1. Motivación: indagar en la necesidad e importancia de la investigación que se pretende realizar.
   2. Objetivos
   3. Estructura del documento
2. Marco teórico
   1. Background: análisis de las tecnologías implicadas
   2. Estado del arte: análisis de la literatura actual, tecnologías, casos recientes, etc.
3. Metodología: definición de la metodología a seguir para esta investigación.
4. Arquitectura: análisis de la arquitectura configurada para la experimentación
5. Iteración X: iteración de experimentos con las distintas estrategias de ataque
   1. Experimento inicial
   2. Análisis
   3. Estrategia
   4. Experimento Final
   5. Análisis de resultados
6. Impacto Social, Legal y Ambiental
7. Conclusiones

Índice ingles

## 1. Introducción

La computación en la nube es una de las tecnologías más importantes de nuestros tiempos, sobre la cual se basan un sinfín de servicios, plataformas, y compañías. Sin embargo, los modelos tradicionales de alquiler de recursos físicos o virtuales en centros de computación suponen un sobrecoste para el contratante en aquellos momentos en los que no se necesitan dichos recursos. Las tecnologías Serverless surgieron hacia finales de los 2000 prometiendo poner fin a esto.

El modelo es simple, para el programador al menos: definir una función abstrayéndose completamente del entorno físico en el que será ejecutada, y pagar única y exclusivamente por el uso de los recursos cuando se ejecute la función. En esto consiste Serverles. En abandonar el servidor tradicional en favor de una función contenerizada, que puede ser desplegada bajo demanda, por solicitud directa o distintos eventos, e igualmente detenerla tras su fin.

Las ventajas son evidentes e incluyen mayor flexibilidad y escalabilidad, mayor facilidad de implementación, disminución de costes, aumento de la rentabilidad de la infraestructura física, portabilidad, etc. Todo esto ha contribuido a una rápida adopción de la tecnología, que, aunque no haya reemplazado a otros modelos cloud, si está en aumento constante con entre el 45% a 75% de los clientes de distintas plataformas Cloud usando al menos una función Serverless[[1]](#footnote-1).

No obstante, una nueva tecnología supone tambien nuevas posibles amenazas. En nuestro caso una de estas amenazas ya se ha tipificado como “Denia lof Wallet” o Denegación de Cartera. Este nuevo tipo de ciberataque pretende generar costes de procesamiento a la victima mediante la activación de las funciones a través de solicitudes directas. Se asemeja al ataque por denegación de servicio, pero a una escala distinta. En este caso no se pretende bloquear el servicio si no que basta con generar costes, ya se de manera puntual o más prolongada en el tiempo.

Se trata de un ataque del que de momento existen pocos ejemplos lo cual proporciona una oportunidad única de investigar y definir estrategias de defensa para prevenirlos antes de que ocurran. En abril de 2024 se hizo público un caso en el que un usuario de AWS vio su factura aumentar más de 1000$ debido a que uno de sus buckets de AWS recibió millones de solicitudes ajenas.[[2]](#footnote-2) En este caso las solicitudes fueron todas en un día, pero otras estrategias permiten el envío de solicitudes en tiempos mucho más prolongados dificultando su detección.

El objetivo de este trabajo es el desarrollo de estrategias de defensas para ataques DoW en entornos serverless. Aprovechando el momento actual, donde no supone una gran amenaza, pretendo investigar el efecto que pueden causar estos ataques, desarrollar estrategias de defensa, y medir su efectividad y aplicabilidad en entornos reales, definiendo una metodología rigurosa y basándome en la literatura científica actual entorno a este fenómeno.

### 1.1 Motivación

Según un informe de Statista sobre el uso de cloud pública, se estima que la facturación de la industria cloud en 2024 alcance la cifra de 773.000M $, con un crecimiento anual estimado de un 18,49% en el periodo 2024-2029.[[3]](#footnote-3) Y es que, debido a la cantidad de ventajas que aporta este tipo de servicios, no es de extrañar. Desde la flexibilidad de la infraestructura, la escalabilidad, hasta la disminución de costes, resulta una opción muy atractiva para una gran cantidad de negocios. Con soluciones personalizadas, distintos tipos de servicios (SaaS, PaaS, IaaS…) y una innovación y mejora constante de la industria, tanto grandes como pequeñas empresas se pueden beneficiar de esta, ya no tan nueva, tecnología.

Esta adopción masiva supone tambien una serie de retos tanto para el cliente como para el proveedor del servicio. En concreto, en un informe elaborado por Flexera donde se incluían un toral de 753 empresas de diversos tamaños, el segundo reto más importante para las entidades es la ciberseguridad.[[4]](#footnote-4) Otro de los retos más importantes es la falta de conocimiento sobre la tecnología, ya que, al subcontratarlo, se delega parcialmente en el proveedor.

Estas amenazas se ven reflejadas en las cifras. Solo en el año 2023 las intrusiones en entornos cloud aumentaron un 75%.[[5]](#footnote-5) Las cifras aumentan tanto en ataques cloud conscientes, donde los actores son conscientes de haberse infiltrado en entornos cloud y utilizan sus características en el ataque, como en ataques agnósticos, donde los actores no son conscientes del entorno cloud. Respectivamente un 110% y un 60%.

Con un panorama político cada vez más tenso, donde tanto organizaciones individuales como otras secundadas por gobiernos hostiles, y donde el cibercrimen supone ya un coste de 9,5B $ anuales[[6]](#footnote-6), es de vital importancia adelantarse a estos actores nefarios y desarrollar estrategias de defensa preventivas para evitar los daños.

### 1.2 Objetivo y alcance

El objetivo de este proyecto es el desarrollo de estrategias de defensa adaptadas a las distintas estrategias de ataque en ciberataques *Denial of Wallet* analizando su efectividad y aplicabilidad en entornos reales, en concreto AWS.

Para ello se pretende desarrollar una arquitectura, usando soluciones de codigo abierto, que simule de manera verosímil un entorno cloud Serverless. En este entorno se simularán ciberataques siguiendo distintas estrategias, se analizarán los ataques determinando los costes incurridos por la víctima, el consumo de recursos virtuales y físicos, y demás aspectos relevantes.

Tras este análisis, se desarrollarán estrategias de defensa adaptadas a cada ataque que permitan reducir el daño causado, analizando su efectividad mediante una comparativa de los daños causados por el mismo ataque antes y después de la implementación de la defensa.

Se analizarán distintas estrategias de ataque que incluyen ataques masivos, similares a DoS, ataques de larga duración, ataques de tasa adaptiva, etc. Para cada ataque la simulación se adaptará a las limitaciones impuestas por los recursos disponibles, utilizando métodos de extrapolación cuando se alcance el límite de los recursos.

Además, se evaluará la aplicabilidad de estas estrategias de defensa en un entorno AWS, determinando si la estrategia puede ser implementada por el usuario o si, por el contrario, recae sobre el dominio del proveedor.

Por último y debido a la importancia del aspecto legal, ético, y social de cualquier labor, se determinarán los impactos sociales y ambientales y se analizarán las implicaciones legales del propio proyecto.

### 1.3 Estructura del documento

Habiendo introducido el proyecto y establecido claramente la motivación y el objetivo del desarrollo de estrategias de defensa para ataques *DoW*, se define en esta sección la estructura del resto del documento.

* Marco Teórico: se realizará un análisis de las tecnologías implicadas en este proyecto y la literatura científica reciente relevante.
* Metodología: se definirá el método de investigación a seguir que permitirá obtener resultados claros y realizar comparativas justas sobre las distintas estrategias obtenidas.
* Arquitectura: se explicará la arquitectura diseñada explicando razonadamente las decisiones de diseño tomadas y sus implicaciones.
* Iteraciones: esta sección contiene la investigación realizada de manera iterativa. Incluirá los experimentos realizados, sus consecuencias, las estrategias obtenidas, y el análisis de los resultados para cada tipo de ataque.
* Impacto social, ambiental, y legal: se realizará un análisis de los distintos aspectos sociales del proyecto.
* Conclusiones: se determinará aquí si los objetivos del proyecto han sido alcanzados y se resumirán las conclusiones obtenidas durante la investigación.

## 2. Marco Teórico

En esta sección se realizará un análisis de las tecnologías implicadas en este proyecto y de la literatura científica relevante.

### 2.1 Serverless y sus tecnologías

El primer ejemplo de tecnologías similares a Serverless se encuentra en Google App Engine (GAE). En el blog del lanzamiento de su versión de prueba, Paul McDonald escribía “*Google App Engine packages these building blocks and takes care of the infrastructure stack, leaving you more time to focus on writing code and improving your application*.” [[7]](#footnote-7) Se puede aquí observar una de las características principales del paradigma Serverless: la abstracción total de la infraestructura física. Por aquel entonces no se utilizaba todavía el término serverless y GAE no ofrecía todas las características de lo que hoy concebimos como Serverless, pudiendo considerarse un precursor.

No fue hasta 2014, con el lanzamiento de AWS Lambda, que se vislumbró por primera vez un producto comercial verdaderamente Serverless. En la entrada al blog de noticias AWS del lanzamiento de AWS Lambda podemos ver las características principales de los servicios serverless[[8]](#footnote-8):

* Orientado a eventos: “*Lambda will automatically run code in response to modifications to objects uploaded to (…) buckets, messages arriving in (…) streams, or table updates (…)*”
* Abstracción de la infraestructura física: “*Lambda is a zero-administration compute platform.*”
* Pago por tiempo de computación: “*You pay for compute time in units of 100 milliseconds and you pay for each request.*”

A medida que se afianzó esta tecnología, surgió el termino FaaS o *Function as a Service* que puede entenderse como un subconjunto de Serverless. Donde Serverless se puede definir como un modelo de computación en la nube en el que se abstrae la infraestructura física, delegando su gestión en el proveedor, y se paga por tiempo de computación, FaaS consiste especificamente en funciones serverless generalmente activadas por eventos externos o internos al entorno cloud. Así FaaS es un tipo de servicio Serverless, pero no todos los servicios Serverless son FaaS.

Los principales proveedores de servicios cloud, Amazon Web Services, Microsoft Azure, y Google Cloud, ofrecen todos sus respectivos productos Serverless: AWS Lambda, Azure Serverless Compute y Google Cloud Run. Sin embargo, para explicar el funcionamiento de este framework nos basaremos en OpenFaaS. Se trata de una solución de codigo abierto que ofrece servicios FaaS sin coste y usando tecnologías estandarizadas y libres. Aunque cada plataforma tenga matices y use tecnologías específicas, y quizas ligeramente distintas, el comportamiento y las tecnologías de todas son similares a grandes rasgos.

#### 2.1.1 Flujo General

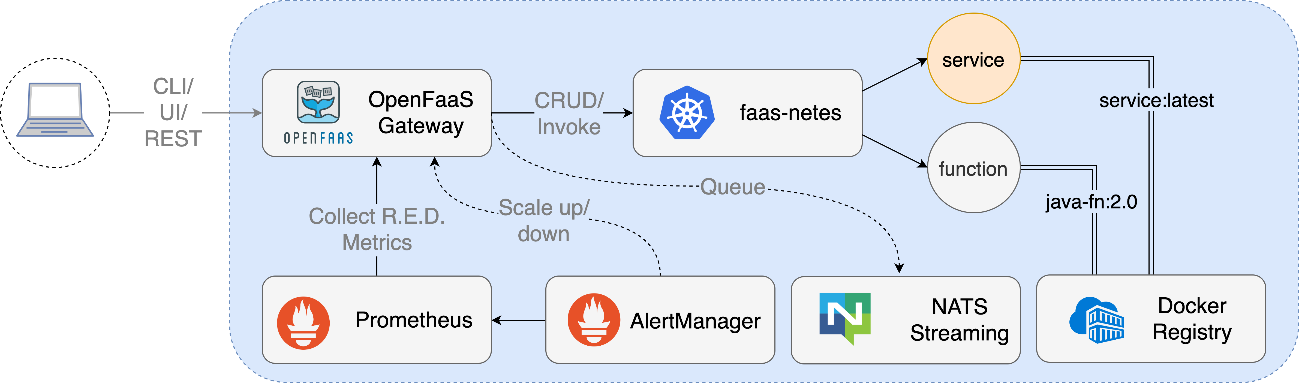
Para indagar en el funcionamiento de OpenFaaS, se explorará primero el flujo de trabajo general definido para una solicitud a una función.

Ilustración 1: Flujo de trabajo conceptual

La función puede ser invocada por consola, interfaz gráfica, o solicitud HTTP. La dirección de destino de esta solicitud seguirá el siguiente formato[[9]](#footnote-9):

http://<dominio>:<puerto>/function/<nombre>

El dominio y puerto dependerán de las preferencias del usuario. Para el dominio puede usarse un dominio DNS o una dirección IP. El puerto se definide cuando se activa el redireccionamiento de puerto al entorno virtual ya que todos los elementos de la ilustración 1 contenidos en el recuadro azul se despliegan sobre un clúster de kubernetes, como veremos más adelante.

Cuando el host recibe la solicitud, lo hace mediante el Gateway API. Este Gateway es el encargado de enviar las invocaciones al operador o a la cola, exponer las métricas recolectadas por prometheus, ofrecer una interfaz gráfica, coordinar la gestión de funciones con el operador, y de manera opcional realiza tareas de balance de cargas.[[10]](#footnote-10)

El siguiente elemento en el flujo es el operador faas-netes. En el caso de la versión pro, este elemento se encarga de la gestión del ciclo de vida de las funciones, escalamiento, balance de cargas, etc. En definitiva, capacita el clúster para ofrecer FaaS. Sin embargo, en la Comunity Edition (CE) esta labor la realiza directamente el API Gateway.[[11]](#footnote-11) [[12]](#footnote-12)

Los dos siguientes elementos son el servicio y la función. La función se encapsula en un POD de kubernetes que contiene el watchdog y la función. El watchdog permite abstraer la parte http del proceso y la función realiza el procesamiento en sí. [[13]](#footnote-13) Su funcionamiento será desglosado con mayor detalle más adelante. Según la documentación oficial de OpenFaaS “*A Kubernetes Service object is also created and is used to access the function's HTTP endpoint on port 80, within the cluster*.”[[14]](#footnote-14)

Hasta ahora se ha examinado un flujo síncrono. Sin embargo, si el flujo fuer asíncrono, la solicitud se enviaría a la cola NATS en vez del operador o trabajador.[[15]](#footnote-15) En este caso el Gateway serializa la solicitud y la inserta en la cola Nats. Otro elemento denominado queue worker se encarga de deserializarla y enviarla al trabajador.  
Como se ha mencionado anteriormente, una de las responsabilidades del Gateway es la exposición de métricas. Estas métricas y otras generadas por los distintos componentes del sistema, son recolectadas por la instancia de Prometheus que se despliega de forma nativa con OpenFaaS CE, y expuestas en formato prometheus.

### 2.2 Estado del Arte

Experimento 1: Tiempo de procesamiento.

Para determinar el efecto que tiene el tamaño de la imagen en el tiempo de procesamiento se realizó el siguiente experimento: una misma imagen, escalada a tamaños que corresponden con incrementos de 10% del tamaño original (10%-100%) fue procesada por la función. Para cada tamaño de la imagen se probó con formato JPG y PNG. En cada iteración se realizaron 31 procesamientos para poder obtener medias significativas. Se anotaron en un archivo CSV los momentos de envió de la solicitud y de respuesta para determinar el tiempo de procesamiento desde el cliente, y se procesaron los datos usando MatLab obteniendo los siguientes gráficos:

Para ambos formatos la relación entre el tamaño de la imagen y el tiempo de procesamiento parece seguir una relación cuadrática, notándose un tiempo de procesamiento significativamente mayor para el formato JPG.

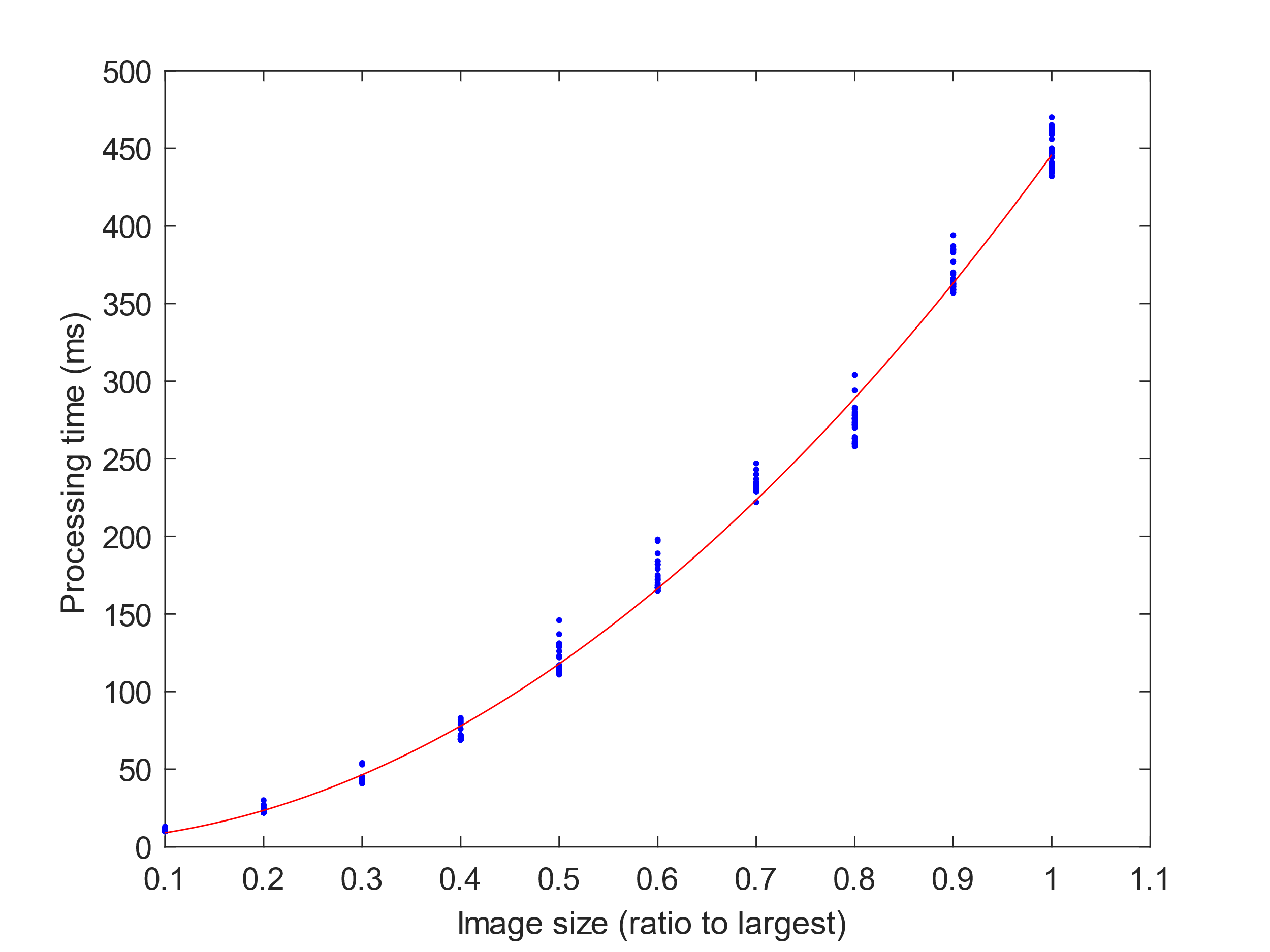


Figure 1: Processing time in relation to image size for PNG format

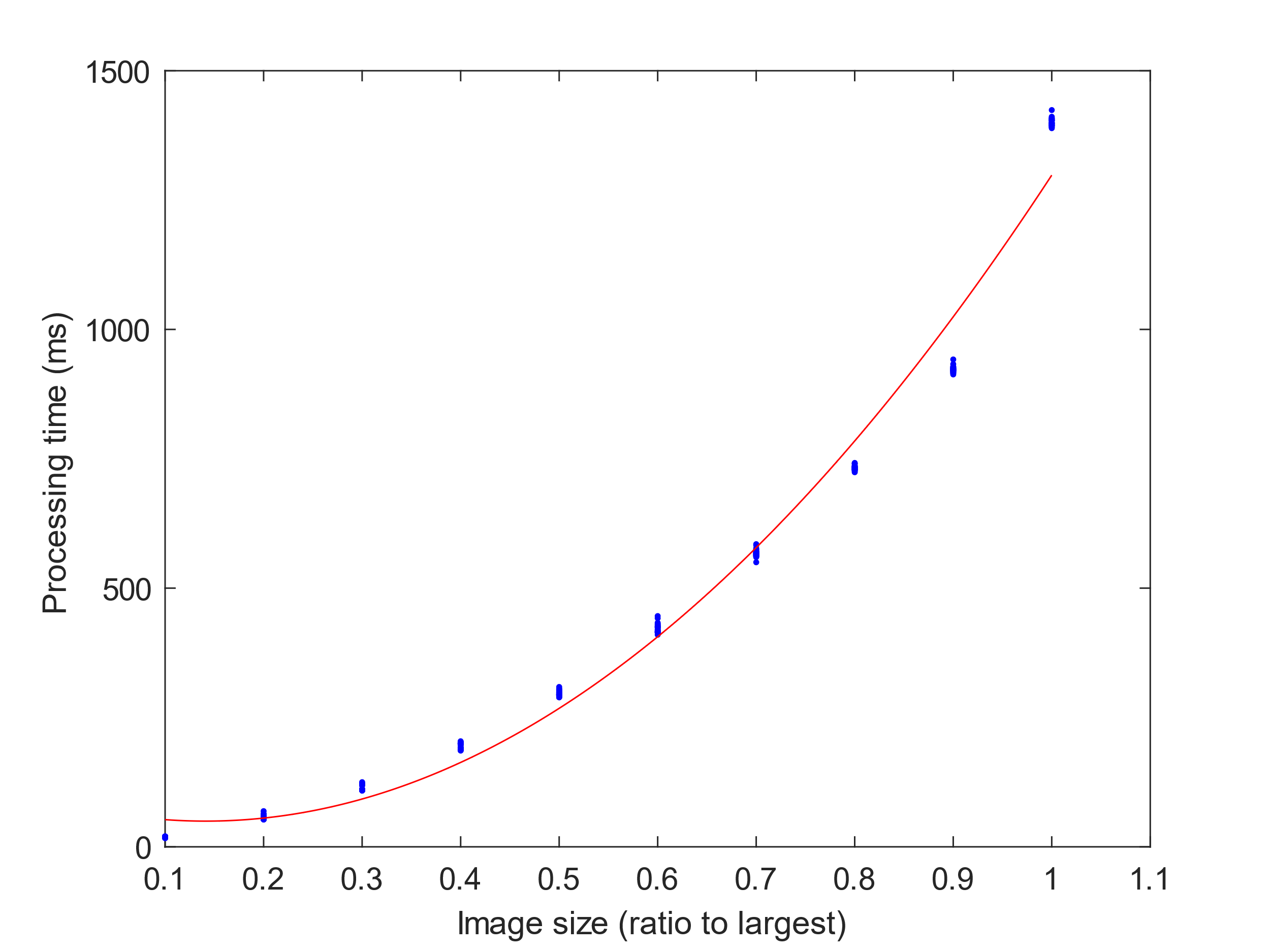


Figure 2: Processing time in relation to image size for JPG format

Experimento 2:

Para determinar que tamaños son mejores para sobrecargar la función se realizó el siguiente experimento. De nuevo con la misma imagen escalada en intervalos de 10% y en formatos JPG y PNG, se realizaron ataques “masivos” (enviar tantas solicitudes como sea posible) que duraron 2 minutos por tamaño y formato. Para cada combinación tamaño-formato se creo un ThreadPool con un numero variable de hilos, que ejecutaban cada una un ataque masivo.

En el dispositivo víctima se recolectaron mediante Grafana las métricas de tiempo medio de procesamiento, número de réplicas de la función, consumo de memoria y consumo de CPU. En el atacante se registro por consola el momento en el que empieza el ataque para cada pareja formato-tamaño, el id de cada hilo, el número de solicitudes enviadas, y el número de respuestas recibidas con código 200.

Debido a que el tiempo de procesamiento para los tamaños es distinto, el número de hilos tuvo que ser ajustado para que no hubiera problemas con la conexión http debido a posibles timeouts. En la Figura X se puede observar el numero de hilos para cada tamaño:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tamaño de la imagen con respecto a la original (%) | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| Número de hilos | 30 | 30 | 30 | 30 | 30 | 15 | 15 | 15 | 15 | 10 |

1. <https://www.datadoghq.com/state-of-serverless/> consultado el 23/09/2024 [↑](#footnote-ref-1)
2. <https://medium.com/@maciej.pocwierz/how-an-empty-s3-bucket-can-make-your-aws-bill-explode-934a383cb8b1> consultado el 23/09/2024 [↑](#footnote-ref-2)
3. <https://www.statista.com/outlook/tmo/public-cloud/worldwide?currency=USD> consultado el 24/09/2024 [↑](#footnote-ref-3)
4. <https://info.flexera.com/CM-REPORT-State-of-the-Cloud-2024-Thanks#challenges> consultado el 24/09/2024 [↑](#footnote-ref-4)
5. <https://www.crowdstrike.com/global-threat-report/> consultado el 24/09/2024 [↑](#footnote-ref-5)
6. <https://www.esentire.com/resources/library/2023-official-cybercrime-report> consultado el 24/09/2024 [↑](#footnote-ref-6)
7. “Introducing Google App Engine + our new blog”, Paul McDonald. Publicado el 07/04/2008. <https://googleappengine.blogspot.com/2008/04/introducing-google-app-engine-our-new.html> consultado el 25/09/2024 [↑](#footnote-ref-7)
8. “AWS Lambda – Run Code in the Cloud”, Jeff Barr. Publicado el 13/11/2024. <https://aws.amazon.com/blogs/aws/run-code-cloud/> consultado el 25/09/2024 [↑](#footnote-ref-8)
9. <https://docs.openfaas.com/architecture/invocations/> consultado el 01/10/2024 [↑](#footnote-ref-9)
10. <https://github.com/openfaas/faas/blob/master/gateway/README.md> consultado el 1/10/2024 [↑](#footnote-ref-10)
11. <https://www.openfaas.com/blog/upgrade-to-the-function-crd/> consultado el 1/10/2024 [↑](#footnote-ref-11)
12. <https://www.openfaas.com/blog/kubernetes-operator-crd/> consultado el 1/10/2024 [↑](#footnote-ref-12)
13. <https://github.com/openfaas/classic-watchdog/blob/master/README.md> consultado el 1/10/2024 [↑](#footnote-ref-13)
14. <https://docs.openfaas.com/architecture/invocations/> consultado el 1/10/2024 [↑](#footnote-ref-14)
15. <https://docs.openfaas.com/reference/async/> consultado el 1/10/2024 [↑](#footnote-ref-15)