Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Despliegue de una plataforma de Serverless Computing

Autor: Juan Parada Claro

Tutor: Francisco José Fernández Jiménez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Despliegue de una plataforma de Serverless Computing

Autor:

Juan Parada Claro

Tutor:

Francisco José Fernández Jiménez

Profesor Colaborador

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado: Despliegue de una plataforma de Serverless Computing

Autor: Juan Parada Claro
Tutor: Francisco José Fernández Jiménez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Este proyecto ha sido posible gracias al inestimable apoyo de mis amigos del grado y en especial a Javier Ros, ya que nos hemos ayudado mutuamente en la compresión de este nuevo mundo para nosotros llamado LaTeXy además compartimos trabajo en el que aprendí algo sobre la tecnología presentada en este proyecto en su vertiente remota.

Por supuesto, también he de agradecer a mis familiares más cercanos que me han ayudado, durante la realización de este proyecto, a seguir adelante y no perder la motivación cuando aparecían los problemas más graves y extenuantes.

No hay que olvidar el trabajo del tutor, siempre apretando e intentando sacar lo mejor de mí y por supuesto intentando ayudar en lo que podía aunque estuviese algo atareado en algunos momentos.

También quiero lanzar mis agradecimientos a los profesores del grado, unos mejores y otros peores, pero siempre intentando enseñar a sus alumnos lo mejor que podían y a mí en particular me han enseñado muchas lecciones de forma general.

Por último, y no menos importante, dar las gracias a las personas que en general han creído en mí y que me han apoyado a seguir adelante con los estudios superiores. Gracias a ellos, he llegado hasta aquí.

Resumen

La computación en línea cada vez es más importante para la sociedad, debido a que cualquier tipo de empresa que quiera dar servicios de forma mundial e incluso en su zona local (para mejorar la atención al cliente) necesita disponer de una forma de ofrecer, al menos, información rápida para que el usuario los pueda encontrar físicamente.

Lógicamente, la mayoría de pequeñas empresas no tienen departamentos de Informática o Telemática para gestionar este tipo de problemas y configurar, por ejemplo, un servidor web propio físico para una pequeña web informativa. Por ello, desde 2006, las empresas que tenían enormes granjas de servidores, decidieron empezar a vender su capacidad de cómputo o almacenamiento sobrante a las empresas que lo necesitaran [1].

A partir de esta situación, en 2012, se nombra por primera vez el concepto de *Serverless Computing* y se empiezan a ver plataformas de procesamiento en la nube que lo incluyen entre sus servicios. Esta tecnología presenta diversas ventajas, aunque no está normalizada por ahora, como el pago por el tiempo real de ejecución de código o la ausencia de mantenimiento de servidores por parte del consumidor.

Esta tecnología se adecúa muy bien también a las nubes privadas de computación basadas en contenedores, debido a que el lanzamiento de estas entidades virtuales casa perfectamente con uno de los conceptos básicos de *Serverless Computing*, el de consumir capacidad de computación en relación con la demanda y así no cargar sin motivo el servidor que hay detrás. Además, cabe recalcar, que esto beneficiará el rendimiento del servidor en tareas que siempre que tengan que estar funcionando [2].

El proyecto consiste en un acercamiento a la ciencia del *Serverless Computing*. En un principio y teóricamente, a las plataformas públicas y posteriormente, a las plataformas instalables privadas de una forma práctica (se instalarán y configurarán). Se tratarán temas clave para la tecnología cómo el auto-escalado y su presente documentación. También, se hará un análisis experimental de rendimiento a cada herramienta. Finalmente, se expondrán opiniones sobre las diversas plataformas utilizadas y se tratará de discernir cuál es mejor de manera subjetiva y objetiva.

Abstract

Online computing is progressively more important for society, it is due to any type of company which wants to give their services worldwide (to improve client attention) needs, at least, fast information for users could find them physically.

Logically, most little companies don't have an especific department of Informatics or Telematics to handle this kind of problems and to configurate, for example, an own web server for a small informative web. Therefore, since 2006, companies which had enormous server farms, they decided to start selling their computing ability or their excess storage to companies which need it.

From this condition, in 2012, Serverless Computing concept is named for first time and cloud processing plataforms began to include in their services. This tecnology presents some benefits, like payment only for real time code execution or lack of mantenaince of servers by the customer.

This technology adapts really well to private computing clouds based in containers, it is due to the launch of this virtual entities joins perfectly with a basic concept of Serverless Computing, to consume computation capacity in relationship with the demand and thus not to charge without reason the server which there is behind. Furthermore, it's worth say, it will benefit the performance of the server in jobs which always it have to work.

The proyect consists in an approach to Serverless Computing science. At the beginning and theoretically, to public plataforms and later, to instalable plataforms in a practical way (it will install and it will configure). It will treat key themes for the technologhy like autoscaling and its actual documentation. Also, it will do an experimental analisis of performance to each tool. Finally, it will expose opinions about the differents used plataforms and it will treat to discern which is better subjectively and objectively.

Índice Abreviado

Agradecimientos I
Resumen III
Abstract V
Índice Abreviado VII

1 Introducción **1**
1.1 Pinceladas sobre *Serverless Computing* 1
1.2 Motivación 2
1.3 Objetivos del trabajo 2
1.4 Metodología 2
1.5 Contenido del trabajo 3

2 Concepto de *Serverless Computing* **5**
2.1 ¿Qué es *Serverless Computing*? 5
2.2 Profundización en *BaaS (Backend as a Service)* 8
2.3 Profundización en *FaaS (Functions as a Service)* 9
2.4 Cloud Native Computing Foundation (CNCF) 10

3 Estado del Arte **13**
3.1 Plataformas remotas o públicas 13
3.2 Keda: una herramienta usada por algunas plataformas instalables *Serverless* 16
3.3 Plataformas instalables o privadas 18

4 Instalación y configuración de varias herramientas *Serverless Computing* para uso privado **23**
4.1 Entorno de trabajo 23
4.2 Instalación de Knative del CNCF 23
4.3 Instalación de OpenWhisk de Apache 26
4.4 Instalación de OpenFaaS 31
4.5 Instalación de OpenFunction de QingCloud 33

5 Creación de una función en varias herramientas de *Serverless Computing* para uso privado **39**
5.1 Características de la función de prueba 39
5.2 Creación de una función en Knative 41
5.3 Creación de una función en OpenWhisk 45
5.4 Creación de una función en OpenFaaS 48
5.5 Creación de una función en OpenFunction 51

6 Configuración del auto-escalado y pruebas de rendimiento sobre las herramientas de *Serverless Computing* para uso privado **55**

6.1 Generalidades en la configuración y metodología de realización de las pruebas 55
6.2 Configuración específica necesaria por herramienta 60
6.3 Resultados obtenidos por las pruebas 62

7 Experiencias, vivencias y evaluación sobre *Serverless Computing* para uso privado 67
7.1 Problemas encontrados durante la realización del trabajo 67
7.2 Comentarios sobre los problemas con la documentación de las herramientas 69
7.3 Comparativa nube privada contra nube pública 70
7.4 Opinión final de cada herramienta 71

8 Conclusiones sobre el proyecto 73
8.1 Conclusiones 73
8.2 Líneas futuras de trabajo 74

Índice de Figuras 77
Índice de Tablas 79
Índice de Códigos 81
Bibliografía 83

Índice

Agradecimientos I
Resumen III
Abstract V
Índice Abreviado VII

1 Introducción **1**
1.1 Pinceladas sobre *Serverless Computing* 1
1.2 Motivación 2
1.3 Objetivos del trabajo 2
1.4 Metodología 2
1.5 Contenido del trabajo 3

2 Concepto de *Serverless Computing* **5**
2.1 ¿Qué es *Serverless Computing*? 5
2.1.1 Beneficios generales 5
2.1.2 Desventajas generales 6
2.1.3 Comparativa con el *Cloud Computing* tradicional 7
Ventajas y desventajas de *CaaS* 7
Ventajas y desventajas de *PaaS* 8
2.2 Profundización en *BaaS (Backend as a Service)* 8
2.2.1 Definición 8
2.2.2 Casos de uso 8
2.3 Profundización en *FaaS (Functions as a Service)* 9
2.3.1 Definición 9
2.3.2 Casos de uso 10
2.4 Cloud Native Computing Foundation (CNCF) 10
2.4.1 ¿Qué es? 11
2.4.2 Importancia en el desarrollo del *cloud* 11
2.4.3 Listado de compañías que la forman 12

3 Estado del Arte **13**
3.1 Plataformas remotas o públicas 13
3.1.1 Funcionalidades básicas 14
AWS Lambda 14
Azure Functions 14
Google Cloud Functions 15
3.2 Keda: una herramienta usada por algunas plataformas instalables *Serverless* 16
3.2.1 Conceptos necesarios para entender Keda 16
3.2.2 Características principales de la herramienta 16
3.3 Plataformas instalables o privadas 18

3.3.1 Knative 19
3.3.2 OpenWhisk 19
3.3.3 OpenFaaS 20
3.3.4 OpenFunction 21

4 Instalación y configuración de varias herramientas *Serverless Computing* para uso privado 23
4.1 Entorno de trabajo 23
4.2 Instalación de Knative del CNCF 23
4.2.1 Requisitos previos 23
Instalación de *kind* 24
Instalación de *kubectl* 24
Instalación de *kn* 25
4.2.2 Instalación 25
4.3 Instalación de OpenWhisk de Apache 26
4.3.1 Requisitos previos 26
4.3.2 Instalación 26
Instalación de *wsk* 26
Instalación de *wskdeploy* 27
Creación de un cluster de *Kubernetes* adaptado para OpenWhisk 27
4.3.3 Configuración 28
4.4 Instalación de OpenFaaS 31
4.4.1 Requisitos previos 31
4.4.2 Instalación 32
Instalación de *faas-cli* 32
Instalación de *OpenFaaS Chart* 32
4.4.3 Configuración 32
4.5 Instalación de OpenFunction de QingCloud 33
4.5.1 Requisitos previos 33
4.5.2 Instalación 35
4.5.3 Configuración 35

5 Creación de una función en varias herramientas de *Serverless Computing* para uso privado 39
5.1 Características de la función de prueba 39
5.1.1 Lenguaje de programación 39
5.1.2 Funcionamiento 39
5.1.3 Pruebas básicas a la función 41
5.2 Creación de una función en Knative 41
Lanzamiento del contenedor 43
Aplicar los valores de configuración 44
Comprobaciones de lanzamiento 44
5.3 Creación de una función en OpenWhisk 45
5.4 Creación de una función en OpenFaaS 48
5.5 Creación de una función en OpenFunction 51

6 Configuración del auto-escalado y pruebas de rendimiento sobre las herramientas de *Serverless Computing* para uso privado 55
6.1 Generalidades en la configuración y metodología de realización de las pruebas 55
6.1.1 Metodología de realización de las pruebas 55
Prueba de carga o de tiempo total de ejecución con muchas peticiones 55
Prueba de velocidad desde cero 57
Prueba de velocidad con función establecida 57
Script completo de prueba 58
6.1.2 Generalidades en la configuración 59
6.2 Configuración específica necesaria por herramienta 60
6.2.1 Knative 60

6.2.2 OpenWhisk 61
6.2.3 OpenFaaS 61
6.2.4 OpenFunction 61
6.3 Resultados obtenidos por las pruebas 62
6.3.1 Prueba de carga 63
6.3.2 Prueba de velocidad desde cero 63
6.3.3 Prueba de velocidad con función establecida 64
6.3.4 Comparativa directa entre herramientas 64

7 Experiencias, vivencias y evaluación sobre *Serverless Computing* para uso privado 67
7.1 Problemas encontrados durante la realización del trabajo 67
7.1.1 Problemas encontrados en la parte teórica 67
7.1.2 Problemas encontrados en la parte práctica 68
Knative 68
OpenWhisk 68
OpenFaaS 69
OpenFunction 69
7.2 Comentarios sobre los problemas con la documentación de las herramientas 69
7.2.1 Knative 69
7.2.2 OpenWhisk 69
7.2.3 OpenFaaS 70
7.2.4 OpenFunction 70
7.3 Comparativa nube privada contra nube pública 70
7.4 Opinión final de cada herramienta 71
7.4.1 Knative 71
7.4.2 OpenWhisk 71
7.4.3 OpenFaaS 71
7.4.4 OpenFunction 72

8 Conclusiones sobre el proyecto 73
8.1 Conclusiones 73
8.2 Líneas futuras de trabajo 74

Índice de Figuras 77
Índice de Tablas 79
Índice de Códigos 81
Bibliografía 83

1 Introducción

Una buena introducción siempre es importante para contextualizar y captar la atención de cualquier tipo de lector.

En este Capítulo se buscará comentar en qué se basa este proyecto de fin de grado. En un principio se introducirán algunos conceptos breves sobre *Serverless Computing* y se presentará en que formas se puede presentar.

A continuación, se comentarán las motivaciones subjetivas y objetivas que han propiciado la selección de este trabajo. Se marcarán unas metas u objetivos para el trabajo y éstos se revisarán al final para ver si han sido cumplidos.

Además, se tratará de explicar la metodología y el orden de trabajo llevado durante la realización de este proyecto. Por último, se comentará en que consistirá el contenido bruto del proyecto, que conceptos se analizarán y que trabajos se realizarán.

1.1 Pinceladas sobre *Serverless Computing*

El *Serverless Computing* es una tecnología de computación en la nube que basa su existencia en simplificar el proceso de desarrollo de aplicaciones distribuidas. Ésto se consigue gracias a que permite al programador desentenderse casi completamente de tareas de configuración de los servidores que alojarán sus diferentes aplicaciones [2].

La ciencia nace, de forma popular, del seno de *Amazon Web Services (AWS)* con la salida al mercado de AWS Lambda en 2012 (en fase de pruebas). Aumenta en popularidad con la creación de las plataformas de otros proveedores de computación en la nube populares, tales como Microsoft o Google. Alcanza el estado actual con la aparición de las primeras herramientas instalables: OpenWhisk de Apache y OpenFaaS (herramienta comunitaria).

La tecnología de *Serverless Computing* ofrece, a mi parecer, una ventaja clara para el usuario y otra para el proveedor:

- Para el usuario: la capacidad de desentenderse completamente de la gestión de la máquina y centrarse en el desarrollo del código a lanzar ahorrando así tiempo.
- Para el proveedor: la habilidad de balancear el uso real de sus servidores y asignar recursos dependiendo del estado del programa a ejecutar y su demanda. Esto permite a la compañía cobrar de manera más satisfactoria al cliente.

Señalar de forma breve que habrá dos formas de enfocar el desarrollo de infraestructura mediante *Serverless Computing*:

- *Functions-as-a-Service (FaaS)*: de esta forma se programarán funciones que serán desencadenadas (lanzadas en los servidores del proveedor) por diversos eventos (peticiones HTTP, otras funciones, eventos en una base de datos...).

- *Backend-as-a-Service (BaaS)*: de esta manera se programará el *backend* de una aplicación que será lanzado por una API REST o HTTP y que nos devolverá un resultado útil.

Un caso de uso de la tecnología sería el de aplicaciones adaptables a una API REST. Por ejemplo, una aplicación distribuida que consista en una agenda de tareas que recuerden automáticamente al usuario determinados eventos programados. En esta aplicación, se tendría un cliente que enviaría la información a guardar a las diversas funciones y éstas procesarían las peticiones recibidas guardando la información en una base de datos asociada. Una aplicación *BaaS* se dedicaría a revisar cuándo habría que recordar y recordará a los diversos usuarios de la aplicación sus diversos eventos programados. Éstas funciones sólo se ejecutarán cuando sean requeridas para que así no se pierda rendimiento en el servidor donde se ejecuten.

Otro caso de uso, que es el que además se va a usar en este proyecto, es el del procesamiento de recursos multimedia. En este trabajo se implementará un algoritmo que cuando se le envíe una imagen JPEG con un fondo blanco, la función eliminará este fondo y convertirá la imagen en un PNG con fondo transparente.

1.2 Motivación

Mi interés por la computación en la nube ha ido en aumento año tras año. Los proveedores de las mismas son muy inteligentes y se anuncian bien en cualquier tipo de evento que exista, sobre todo deportivos. Este factor puede parecer mínimo, pero, los anuncios siempre acaban calando de una forma u otra en el receptor.

Es necesario recalcar que era muy probable que me encontrase con ellas, ya que soy un apasionado de todo lo que está detrás, enormes granjas de servidores, gran cantidad de ingeniería en su trasfondo, etc.

En primero de Bachillerato, tuve la oportunidad de encontrarme con algo similar pero a menor escala gracias al profesor de Informática del instituto. Tuvimos la ocasión de hacer una excursión al Centro Informático Científico de Andalucía (CICA), situado en la zona universitaria de Reina Mercedes. En esta visita nos explicaron bastantes cosas sobre los servidores que tenían allí y esto hizo que me entrase bastante curiosidad.

Para rematar mis encuentros con el *Cloud Computing*, este mismo año he cursado la asignatura del grado *Servicios Telemáticos Avanzados*, la cuál tiene un trabajo de curso que elegí hacerlo sobre *Terraform IaC (Infrastructure as Code)*, una herramienta que nos permite, entre otras cosas, generar máquinas virtuales en los servidores de los proveedores típicos en la nube.

En cuanto a motivaciones no tan personales, se intentará analizar en que situación de desarrollo se encuentra el *Serveless Computing*, qué papel juega en cuanto a la computación en la nube tradicional y qué puede aportar esta tecnología a un desarrollador de aplicaciones distribuidas que quiere probar nuevas formas de programar sus desarrollos (dejando de lado los servidores tradicionales).

Todos los hechos presentados anteriormente hicieron que me decantase por la realización del tema que conlleva este proyecto de fin de grado.

1.3 Objetivos del trabajo

Cualquier proyecto académico de este tipo debe presentar unas metas que se traten de cumplir. Esto es así porque un alumno que lo haya realizado debe sentir que ha aprendido nuevos conceptos y que el trabajo hecho le ha servido para progresar en sus conocimientos personales.

Esta sección ha de ser importante también para el lector, esto es así porque la persona que lea este proyecto debe aprender, aunque al menos sea en parte, de los objetivos presentados a continuación:

- Comprender el concepto de *Serverless Computing* y entender las formas que presenta.
- Conocer las plataformas de esta tecnología tanto en uso privado y como en uso público.
- Comprender el funcionamiento de las plataformas privadas.
- Ser capaz de instalar, configurar y realizar una implementación básica de las diversas herramientas analizadas.
- Conocer el concepto de auto-escalado y saber configurar las herramientas para que pueda ser usado.
- Analizar de manera objetiva qué herramienta instalable es la mejor tras haber realizado una serie de pruebas.

1.4 Metodología

Este trabajo se ha realizado siguiendo una forma de proceder muy típica en la realización de este tipo de proyectos. El orden y división de trabajo ha sido el que se presenta a continuación:

1. Búsqueda de información y bibliografía básica usada.
2. Redacción de introducciones y resúmenes.
3. Redacción de los capítulos teóricos.
4. Realización de la implementación (instalación, configuración y puesta en funcionamiento de una pequeña función) en las herramientas.
5. Realización de la configuración para el auto-escalado y de la creación de las pruebas.
6. Paso de pruebas a todas las herramientas.
7. Redacción y creación de grafismos sobre los resultados de las pruebas.
8. Redacción del capítulo de experiencias y de la evaluación subjetiva de las herramientas.
9. Redacción de conclusiones.

En cuanto a la redacción de los capítulos prácticos, ésta se ha hecho en paralelo con respecto a la propia realización de la parte práctica del proyecto. En cuanto a la bibliografía, se buscó una inicial para los contenidos teóricos y se ha ido añadiendo más a medida que se ha ido avanzando en la parte práctica.

1.5 Contenido del trabajo

El trabajo constará de siete capítulos, sin contar este capítulo introductorio. Cómo se usará posteriormente la división que se comentará a continuación, es de recibo aclararla en esta sección del inicio del proyecto. El trabajo de dividirá conceptualmente en dos partes: parte práctica y parte teórica. Estas partes son aludidas dentro del trabajo en varias ocasiones.
A cada parte conceptual se le asociarán los siguientes capítulos:

- **Parte teórica**:
  - Capítulo sobre *Serverless Computing*: se aclarará qué significa *Serverless* y qué beneficios y perjuicios conlleva usarlo, cómo se definen las dos formas que componen esta tecnología y además, se comentará el papel que juega el Cloud Native Computing Foundation (CNCF) en cuanto al desarrollo de la ciencia.
  - Capítulo sobre el Estado del Arte: se conocerán las plataformas remotas más usadas y se desarrollarán en profundidad las plataformas instalables que se usarán en la parte práctica.
  - Capítulo conclusivo: se verán los diferentes saberes que se pueden obtener tras realizar el proyecto.
- **Parte práctica**:
  - Capítulo sobre la instalación, configuración y realización de una implementación básica en las diversas herramientas: se explicará cómo se instalan las diversas herramientas y se realizará una configuración básica.
  - Capítulo sobre la creación de una función básica: se implementará una pequeña función para probar que la herramienta está funcionando de manera correcta.
  - Capítulo sobre la configuración del auto-escalado y sobre la realización de pruebas a las herramientas: se expondrá qué configuración de auto-escalado se usará, se aplicará esta configuración en las distintas herramientas, se comentarán las pruebas a realizar y se presentarán los resultados de las mismas.
  - Capítulo sobre las experiencias vividas y conclusiones subjetivas de las diversas herramientas: se expondrán los problemas percibidos en la parte práctica (destacando los sufridos por la documentación) y se comentará la opinión personal en base a las vivencias de cada herramienta.

2 Concepto de *Serverless Computing*

La simplicidad es la máxima sofisticación.

Leonardo Da Vinci

Este capítulo explica, de forma más detallada, el concepto que lleva tras de sí el *Serverless Computing*, y en qué formas se presenta con una breve visión entre las mismas y alguna comparación con las otras técnicas de hacer computación en la nube, tanto de forma privada como a través de un proveedor de servicios popular.

Además contendrá una sección dedicada a la asociación empresarial que trabaja por mejorar y normalizar, hasta cierto punto, todas las tecnologías relacionadas con la nube. Esta organización es probablemente la que más está impulsando el desarrollo de *Serverless Computing* en la nube privada.

2.1 ¿Qué es *Serverless Computing*?

El concepto de *Serverless Computing* es capaz de englobar varias tecnologías, ya que simplemente se queda en la construcción y ejecución de aplicaciones distribuidas las cuales no van a necesitar de una gestión de los servidores donde son alojadas, ya que esta parte la realizará la tecnología que se esté utilizando, facilitando así el desarrollo de las mismas [2][3].

Este hecho, nos lleva a ver que estas tecnologías son capaces de añadir una capa de abstracción en la cima de la pirámide de la infraestructura de la computación en la nube, que refinan, de manera general, la comodidad hacia el desarrollador de aplicaciones *cloud* [4].

En general, todas las aplicaciones que se lanzan con esta tecnología se basan en eventos, es decir, para que el código que se haya desarrollado se lance, se debe recibir algún tipo de notificación que estemos esperando. Sin este punto, el *Serverless* perdería valor, ya que los beneficios que presenta se verían reducidos en parte [4].

En los siguientes apartados de esta subsección se verán qué beneficios y qué desventajas presenta esta ciencia. Además, se tratará de comparar con los servicios que son algo más longevos en la computación en la nube como *Containers-as-a-Service (CaaS)* o *Platform-as-a-Service (PaaS)*.

2.1.1 Beneficios generales

Si las herramientas que implementan esta tecnología no tuviesen una larga lista de beneficios, éstas no estarían evolucionando de la forma tan rápida que lo hacen (en cuanto a número y calidad de las mismas), siendo una ciencia relativamente reciente. Por ello, en esta subsección, se tratarán de redactar las ventajas que produce usar *Serverless Computing* en el desarrollo y en la gestión de las aplicaciones.

Hay beneficios que son más claros a primera vista, ya que benefician al desarrollador, y son estos [2]:

- **Sin gestión, actualización y abastecimiento**: no será necesario ni configurar, ni preocuparse por que el software sea actual y por supuesto, ni de tener que lanzar los servidores, contenedores o máquinas virtuales que hay detrás de nuestras aplicaciones distribuidas.
- **Escalabilidad flexible**: un producto desarrollado en *FaaS* o en *BaaS* podrá escalar de forma instantánea y precisa para manejar individualmente cada petición entrante. Esta funcionalidad permite al

desarrollador olvidarse de planear la capacidad de cómputo que requiere su desarrollo, además tampoco se necesitará de configurar las reglas de escalado automático y a qué disparadores se asocian. En definitiva, el escalado ocurre sin ningún tipo de intervención del desarrollador (o si acaso con una intervención mínima).

- **Sin coste computacional en reposo**: no hay carga ni económica ni computacional cuando los códigos que tengamos preparados para ser lanzados no reciban ningún tipo de petición. Eso sí, hay que tener en cuenta que esto se aplica a esta tecnología, si por ejemplo se necesitase almacenamiento para la aplicación, no conllevará este beneficio por completo, ya que se estarán referenciando tecnologías diferentes de computación en la nube que si muestran un coste computacional en reposo.

La tecnología presenta otros beneficios que no son tan evidentes como los anteriores y además algunos también engloban al proveedor, en el caso de que haya. Son los siguientes [2][4]:

- Nuevo modelo de negocio en la nube llamado *pay-as-you-go* (paga por lo que usas), que beneficia al cliente en cuanto a su economía y al proveedor en cuanto al uso más eficiente de sus recursos .
- Tiempos de ejecución, sistemas operativos y incluso contenedores utilizados están completamente abstraídos del código, lo que permite una mejor portabilidad entre proveedores o entre herramientas privadas. Además este punto también beneficia al proveedor, ya que puede seleccionar en que máquinas ejecutar tus servicios sin que el cliente tenga ningún reproche debido a que no notará diferencia apreciable.
- Modelo ajustable al emergente mundo de los flujos de trabajo basados en eventos que se están desarrollando actualmente con el surgimiento del Internet de las Cosas (IoT, Internet of Things)
- Tecnología que encaja con el modelo de micro-servicios, de hecho puede ser considerada como una mejora del mismo.
- El coste de escalado es menor para el usuario y proveedor ya que el escalado es más lineal por el hecho de la modularidad presente en esta tecnología en comparación con las otras tecnologías *cloud*. En la Figura 2.1 extraída de [4] se puede ver este fenómeno.

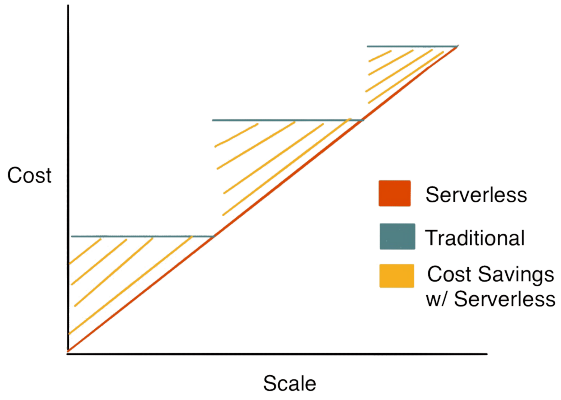

Figura 2.1 Gráfica coste-escalabilidad extraída de [4].

- Debido a la reducción drástica del tiempo gastado en la gestión de servidores, el tiempo desde que se empieza el desarrollo de una aplicación distribuida hasta que es lanzada al mercado es menor, lo cuál podría reducir los costes empleados en la realización de la misma.

2.1.2 Desventajas generales

Cómo cualquier tecnología, es evidente que las herramientas *Serverless Computing* deben tener algún tipo de desventaja, aunque no sean tantas en comparación con las ventajas. Por ende, en esta subsección se listarán y explicarán algunas de ellas, recalcando las más importantes [2][4].

- El simple hecho de ser una **tecnología reciente** y en rápida evolución hace que la documentación sea contradictoria y difícil de comprender, lo cuál hace complicado aprender de forma autónoma los conceptos más profundos de ésta.

- Debido a que la **naturaleza de la ejecución es más dinámica**, puede ser complicado de encontrar los errores que más problemas den, con respecto a *CaaS* y *PaaS*.
- Por la propia **estructura de *pay-as-you-go***, las aplicaciones que se ejecuten con esta tecnología pueden sufrir de pérdidas de rendimiento en los escalados desde 0 (tras un tiempo de inactividad), sobretodo en las plataformas remotas, ya que los proveedores buscarán la máxima eficiencia y tratarán de no asignar recursos al cliente que no esté lanzando ningún código. Recalcar que esto también ocurrirá, aunque en menor medida, en las plataformas instalables ya que buscarán la eficiencia del servidor donde se instalen. Este fenómeno se llama ***cold start***.
- Las diversas herramientas presentan una **falta de estandarización** y además padecen de algo de inmadurez en el ecosistema (por ejemplo, compatibilidad con algunos productores de eventos o falta de compresión de los generadores de eventos propios del proveedor).
- El **control de la infraestructura es mínimo**, únicamente pudiendo cambiar el lenguaje de programación con el que ejecutamos el código. Limita la personalización y el control de métricas de la misma (normalmente sólo tiempo de ejecución o memoria asignada) .
- Las **aplicaciones que tardan mucho tiempo en ejecutarse** no encajan en el modelo de *Serverless Computing*, ya que se perderían los beneficios de la construcción y escalabilidad rápida. Tanto es así, que la mayoría de proveedores limitan el tiempo de ejecución a un máximo de 5 minutos.
- La **infraestructura es compartida** por diferentes consumidores (en plataformas remotas) y por diferentes aplicaciones (en plataformas remotas y privadas). Esto puede provocar situaciones en la que vecinos o aplicaciones que sean muy ruidosos (mucha carga de generación de funciones y *backend*) provoquen pérdida de rendimiento en nuestro código y retrasos de ejecución.
  También podría darse la situación de errores en los que una función de un consumidor evite que una función de otro se ejecute porque se había asignado la misma infraestructura, esto implica errores de seguridad y robustez.

2.1.3 Comparativa con el *Cloud Computing* tradicional

En esta subsección se tratará de comparar las diferentes tecnologías de computación en la nube que pueden hacer un papel parecido al *Serverless Computing* con éste. Esas ciencias del *cloud* son *CaaS* y *PaaS*. Para empezar con este apartado, se definirán las dos a continuación:

- ***Container-as-a-Service (CaaS)* [5]:** es un servicio de computación en la nube que está basado en la gestión y puesta en marcha de aplicaciones que usan la abstracción de contenedor como base para su funcionamiento. El proveedor de este servicio ofrecerá la infraestructura (su capacidad de cómputo), la plataforma usada para generar y gestionar estos contenedores y lanzará la configuración usada. Algunos ejemplos de servicios *CaaS* son *Amazon EC2 Container Service*, *Microsoft Azure Container Service* o *Google Container Engine*.
- ***Plataform-as-a-Service (PaaS)* [6]:** es un servicio de computación en la nube que está basado en la capacidad de dar a los clientes el suministro, instanciado, ejecución y gestión de una plataforma de computación completa de una o más aplicaciones asociadas sin la complejidad de construir y mantener la infraestructura necesaria detrás de la misma. Algunos ejemplos de servicios *PaaS* son *Red Hat OpenShift* o *AWS Elastic Beanstalk*.

Tras contextualizar estas tecnologías, se detallarán a continuación los diferentes beneficios y perjuicios más destacables que reportan en comparación con el *Serverless Computing* más utilizado, el *FaaS*.

Ventajas y desventajas de *CaaS*

1. **Ventajas** [2]:
   a) Máximo control y responsabilidad de cómo se usa la potencia, debido a la elección de la imagen a lanzar del contenedor con una versión, configuración y política de lanzamiento propia.
   b) Posibilidad grande de reutilización y portabilidad de los contenedores usados.
   c) Sencillo llevar aplicaciones basadas en el uso de contenedores a la nube pública.
2. **Desventajas** [2]:
   a) Mayor responsabilidad en la gestión de la potencia, lo que puede llevar al consumidor a gestionar mal los recursos.
   b) Mayor tiempo de gestión de la carga computacional y por tanto del escalado.

c) Normalmente, mayor tiempo de lanzamiento en la nube.
d) Mayor importancia de los servicios de monitorización debido a que en la integración habrá más errores.

Ventajas y desventajas de *PaaS*

1. **Ventajas** [2]:
   a) El marco de desarrollo es el código fuente de la aplicación y los servicios a los que ésta se conecta. Menos control sobre cómo se ejecuta por detrás, pero permite opciones sobre configuración y escalado.
   b) Sin necesidad de gestionar el sistema operativo que se ejecuta detrás.
   c) Los proveedores dan paquetes básicos para construir la plataforma, éstos son ligeramente modificables para dar el control que se quiera sobre la misma.
   d) Esta tecnología encaja muy bien con muchas de las aplicaciones web ya creadas, debido a que presenta un modelo de programación estable y clásico para éstas.
2. **Desventajas** [2]:
   a) Pérdida de algo de control sobre el sistema operativo debido a la existencia de los paquetes por defecto para construcción de plataformas.
   b) Sesgo mayor en cuanto a la estructura de la aplicación, se tiende hacia la arquitectura de microservicios (con mejores prácticas), a costa de perder flexibilidad en la plataforma.
   c) Posible futuro monopolio en cuanto al proveedor, lo que provocaría más dificultades en cuanto a portabilidad.

En la Tabla 2.1 se muestran los contextos en los cuales se usa cada una de las tecnologías con respecto al *Serveless Computing*.

2.2 Profundización en *BaaS (Backend as a Service)*

Está tecnología, también conocida cómo *MBaaS* por su claro y frecuente uso cómo *backend* para aplicaciones móviles [7], sigue un segundo plano con respecto a *FaaS*, ya que su uso, en general, es menor que la tecnología de funciones. Uno de los motivos es el simple hecho de que uno de sus principales fuertes es su combinación con la otra tecnología de *Serverless*. Aún así, sólo una de las compañías grandes de *Cloud Computing*, la implementa junto a *FaaS* bajo el mismo nombre: *Amazon Web Services* con AWS Lambda.

En está sección, se verá, de forma breve, debido a su importancia, de qué trata esta tecnología, sus principales puntos que la hacen interesante y en qué situaciones de uso encaja.

2.2.1 Definición

Para entender la tecnología de forma correcta, es necesario explicar que es el *backend*. Este concepto, se refiere a la parte software de la aplicación o proyecto que estemos desarrollando que se dedica a tareas computacionales puras, o dicho de otra forma la que no se dedica a la parte de la presentación gráfica de la aplicación. Es decir, el *backend*, en este caso será de apoyo, por tanto realizará tareas de cálculo computacional puro de soporte, de acceso a almacenamiento externo o de notificaciones para la dicha aplicación [8].

El concepto de *BaaS* se refiere al servicio *cloud* que nace para dar la capacidad a las aplicaciones que lo usan de centrarse en su campo de negocio . Esta tecnología es capaz de gestionar fuera de la aplicación servicios que son necesarios para su funcionamiento, pero no es lo que va a dar valor añadido al proyecto que se esté desarrollando. Un ejemplo de uso sería el de la gestión de las acciones sobre un servicio de base de datos o almacenamiento remoto que sustenta la información que usa la aplicación [2].

Es necesario recalcar, que *BaaS* juega un papel importante con respecto a la infraestructura completa de *Serverless Computing*. Esto se debe a que presentará un rol que será capaz de ayudar a las propias funciones *FaaS*, ya que descongestionará a éstas de trabajos secundarios. Éstos son importantes, pero no darán una mejora palpable al cliente del servicio que finalmente quedará.

2.2.2 Casos de uso

Está tecnología tiene dos principales casos de uso, uno más habitual que el otro, pero ambos son interesantes desde el punto de vista general de *Serverless Computing*. Éstos son [2][7]:

Tabla 2.1 Recomendaciones de uso: Serverless vs CaaS vs PaaS [2].

Nombre de la tecnología	Contexto de uso recomendado
Serverless Computing (FaaS)	- Desarrolladores que se quieren centrar más en el funcionamiento del negocio gracias a las propiedades de las funciones que lanzan. - Desarrolladores que desean construir sus aplicaciones rápidamente sin preocuparse sobre las aspectos operacionales. - Desarrolladores que buscan crear aplicaciones que respondan en base a eventos específicos como cambios en bases de datos, lecturas de máquinas IoT, o entradas generadas por personas. - Organizaciones que están cómodas en cuanto a usar tecnologías novedosas con poca documentación de calidad y con poco desarrollo en cuanto a las buenas prácticas de las mismas.
CaaS	- Desarrolladores que quieren tener control sobre como su aplicación y sus dependencias (empaquetadas y versionadas), asegurando portabilidad y capacidad de reusarla a través de los diferentes proveedores. - Desarrolladores que buscan un gran rendimiento dentro de un conjunto cohesionado de microservicios interdependientes que sean capaz de escalar de forma independiente. - Organizaciones que quieren mudar sus contenedores de nube privada a nube pública o que quieren ejecutar una parte en su nube privada y otra en la parte pública.
PaaS	- Desarrolladores que desean una plataforma donde lanzar su código y así poder centrarse en el desarrollo de su entorno de aplicación (código fuente y ficheros) sin la necesidad de preocuparse del sistema operativo. - Desarrolladores que quieren crear servicios más clásicos basados en HTTP con nombres de host enrutables por defecto. - Organizaciones que están acomodadas con un modelo más desarrollado de computación en la nube con una documentación avanzada.

- **De soporte para aplicaciones móviles**: permite a los desarrolladores construir un *backend* basado normalmente en API REST que realice tareas secundarias de la índole de:
  - Procesamiento de tareas asíncronas.
  - Envío de notificaciones al usuario.
  - Búsqueda *online* rápida de elementos para la aplicación.
- **De soporte para análisis estadístico**: permite a los desarrolladores incluir una monitorización sencilla de los recursos que consume su aplicación o sobre las acciones realizadas por el usuario que sean interesantes, sin que ésta necesite gastar algo de porcentaje de CPU en hacerlo por si misma.

2.3 Profundización en *FaaS (Functions as a Service)*

Esta ciencia nace de la mano de AWS Lambda en 2012, la cuál es la que la empieza a promocionarla a lo que es hoy, haciéndola una tecnología bastante masificada y con un futuro, con alta probabilidad muy prometedor [3].

En esta sección se verá qué es esta tecnología, las características que la hacen interesante y en qué casos de uso se suele presentar.

2.3.1 Definición

El concepto de *FaaS* se refiere al servicio de computación en la nube, tanto pública como privada, que es capaz de, a partir de unos trozos de código sin estado, llamados funciones, lanzar de forma automática contenedores cuando se reciba algún tipo de evento o disparador. Estos contenedores son efímeros, es decir,

que al acabar la ejecución de la función, serán eliminados hasta nuevo aviso, siendo aprovechable de esta forma el espacio computacional que había sido ocupado por los mismos [3].

Esta ciencia permite al desarrollador montar aplicaciones en las que no hace falta gastar una gran cantidad de tiempo en la configuración y lanzamiento de los contenedores que las soportarán. Esto es debido a que el desarrollador no tendrá posibilidad de elegir muchos parámetros de los contenedores que se lanzarán, únicamente parámetros como el lenguaje de programación usado, el tiempo máximo de ejecución (no podrá exceder los 10 minutos) y en algunas ocasiones, la arquitectura de procesador usada [4].

FaaS puede ser considerado como el "núcleo" de la plataforma de *Serverless Computing*, ya que será la base de aplicaciones completas, basadas en eventos, que quieran disponer de escalado automático y posibilidad de pago por uso exacto de computación [2].

Destacar que los proveedores de *cloud* han aceptado está tecnología con mucho gusto, ya que ésta ha sido capaz de atraer a nuevos clientes que no estaban habituados al uso de la computación en la nube, debido a la facilidad que *FaaS* presenta para programar las aplicaciones en ella y la bajada del gasto de los propios consumidores de mantener y gestionar sus servidores [9].

2.3.2 Casos de uso

En está subsección desarrollaré en que campos de la creación de aplicaciones aplica esta tecnología y con que porcentaje de uso aplican [9].

Recalcar que los casos de uso recomendados generales ya los he desarrollado en la Tabla 2.1 que apareció anteriormente.

Para organizar mejor esta información, he realizado la Tabla 2.2 que aparece a continuación. Es destacable puntualizar que la información de los porcentajes de uso son de un estudio de votaciones sobre usuarios de la tecnología [10].

Tabla 2.2 Ejemplos de uso y su porcentaje.

Caso de uso	¿En qué consiste?	Porcentaje de uso
Aplicaciones Web o que sirvan una API	- Apps Web: página web para que cada contenido responda a una función. - Apps con API: aplicaciones que requieran cierta información o procesamiento en determinados instantes, usando API REST o HTTP.	32 %
Procesamiento de datos	Gestión de cambios en sistemas de almacenamiento o bases de datos.	21 %
Integración como soporte con otros servicios de la nube	Ayuda en ejecución de ciertas partes de aplicaciones, también desarrolladas en la nube, que congenian mejor con la ejecución basada en eventos o con ejecuciones esporádicas.	17 %
Herramienta de soporte para la aplicación	Ayuda en ejecución de ciertas partes de aplicaciones que congenian mejor con la ejecución basada en eventos o con ejecuciones esporádicas.	16 %
Robots para conversaciones	Tanto como hablado o escrito, cuando se pide algo, deciden el código a ejecutar, por tanto encaja bien con el modelo.	8 %
IoT	Cuando las diversas máquinas inteligentes requieren de procesamiento pesado o de comunicarse mediante intermediarios, todo para mejorar su rendimiento.	6 %

2.4 Cloud Native Computing Foundation (CNCF)

En está sección se expondrá, con una visión general, qué es está organización y por qué es importante para el desarrollo de las tecnologías de computación en la nube, especialmente para el *Serverless Computing* [11].

2.4.1 ¿Qué es?

Para comprender el significado de la organización, es necesario comprender el concepto de *Cloud Native* que ellos mismos explican en su página web. Este concepto se refiere a las tecnologías que atraen a organizaciones a construir sus aplicaciones en sistemas dinámicos modernos (nube pública, privada o híbrida) que permiten escalabilidad. Este concepto incluye términos como contenedores, mallas de servicios, micro-servicios, infraestructura inmutable o API declarativas.

Está fundación se podría definir como un conglomerado de empresas asociadas que trabajan, crean servicios o herramientas de computación en la nube. El objetivo que tiene la organización es el de conducir a las diferentes tecnologías a promover y soportar un ecosistema de código abierto y de proyectos comerciales neutrales. Lo hace democratizando los patrones modernos de trabajo, para que las innovaciones sean accesibles para todo el que lo desee.

2.4.2 Importancia en el desarrollo del *cloud*

La importancia de esta asociación reside en su forma de trabajar, sus valores y su rol desempeñado en el desarrollo de las herramientas.

La misión principal es la de hacer de la computación en la nube un campo de estudio más amplio y diverso. Este objetivo se consigue cumpliendo de forma estricta los diversos roles que tiene designados la fundación. Éstos son:

- **Administración de proyectos:** este rol se refiere a asegurar que las tecnologías están disponibles para la comunidad y sin influencia hacia alguna parte o empresa.
  También se deberá asegurar que la marca de la tecnología (logo y distintivo de la misma) es utilizada de forma apropiada por los miembros de la comunidad, con un gran énfasis en una experiencia de usuario uniforme y altos niveles de compatibilidad.
- **Fomentar el crecimiento y evolución del ecosistema**: este rol se refiere a la evaluación de qué tecnologías nuevas deberían ser añadidas, cumpliendo con la visión de la organización, y trabajando para que éstas sean alentadas a que la comunidad las desarrolle e integre, eso sí, si evolucionan con la agenda de la asociación.
  También será necesario proveer una forma de fomentar aspectos técnicos comunes entre las distintas tecnologías.
- **Promocionar tecnologías no tan populares**: este rol se refiere al acercamiento a su definición y configuración. Para conseguirlo, se suelen programar eventos y conferencias en torno a ellas.
- **Servir a la comunidad**: este rol se refiere a tratar de hacer la tecnología más accesible y fiable mediante ofrecer una integración total y una creación cualificada de cada una de las diferentes piezas, con un ritmo acorde a la arquitectura.

Los principios de la fundación definirán con alta probabilidad los valores que presentarán las diversas herramientas en las que la asociación ayude en su desarrollo. Éstos principios son:

- **Rápida:** la asociación permite a los proyectos progresar velozmente para adaptarse a la adopción agresiva de los usuarios.
- **Abierta:** la organización es accesible, además es independiente y opera sin intereses partidistas. Acepta a los contribuidores en base a sus méritos y busca que la comunidad técnica sea transparente.
- **Limpia:** la fundación tratará de evitar las decisiones previo pago y buscará sancionar estos comportamientos.
- **Fuerte en cuanto a su identidad técnica:** el conglomerado tratará de mantener un alto grado en cuanto a su propia identidad tecnológica que ha ido forjando a través de los proyectos realizados.
- **Con barreras claras:** la asociación declarará sus objetivos evidentes y en algunos casos, especificará cuáles son las metas que no se quieren buscar, para así permitir el correcto desarrollo de los proyectos.
- **Escalable:** la organización tratará de soportar a la vez proyectos de distinta índole. Esto implicará que algunos trabajos opcionales no salgan de la fase de desarrollo.
- **Independencia:** la fundación tratará que las especificaciones desarrolladas no estén limitadas en cuanto a la arquitectura o sistema operativo que se quiera implementar.

2.4.3 Listado de compañías que la forman

Para comenzar esta subsección, comentar que el conglomerado empresarial presenta unos rangos de membresía que definirán los derechos y deberes que tendrá la compañía que atesore cada rango. Los rangos de más privilegios a menos son los siguientes: *Platinum, Gold, Silver, End User, Academic* y *Non-Profit*.

En total la organización está compuesta por 813 empresas, compañías o corporaciones. Para ver la lista de las compañías y organizaciones con sus logos puede visitar el siguiente enlace: *https://www.cncf.io/about/members/*.

3 Estado del Arte

Con el conocimiento se acrecientan las dudas.

JOHANN WOLFGANG GOETHE

Las diversas tecnologías basadas en la computación en la nube (privada y pública) llevan en desarrollo un tiempo relativamente breve (desde 2005) comparado con otro tipo de desarrollos del campo de la ingeniería telemática [1].

Por ello, hay que tener en cuenta que la gran mayoría de empresas que invierten en este tipo de innovaciones es porque tienen un gran capital disponible o están en un gran momento de desarrollo [1]. Estas compañías son del tipo *Amazon*, *Google*, *IBM*, *Oracle*... También cabe destacar en estos avances a organizaciones como la *CNCF* (unión de compañias de computación en la nube) o la *Apache Software Foundation* (comunidad de software libre).

Todas estas empresas y organizaciones, a partir de 2012, empezaron a ofrecer también algún tipo de servicio relacionado con el *Serverless Computing* [12]. En este capítulo se analizarán y compararán de forma breve las diferentes implementaciones más populares tanto en nube pública como en nube privada.

También se tratará de analizar cuál de estas implementaciones es más usada entre las empresas o el público general.

3.1 Plataformas remotas o públicas

Esta sección no es en la que se centra el trabajo por tanto describiré las herramientas más comunes y lo haré de forma breve. Es interesante contextualizar de forma sencilla el funcionamiento básico de las mismas ya que tendrán ciertas similitudes con las plataformas instalables.

Comparativa directa básica (lenguajes soportados por defecto, cronología y cuota de mercado aproximada, debido a la poca cantidad de datos sobre ésta) entre las diferentes herramientas con la siguiente tabla:

Tabla 3.1 Comparativa plataformas remotas.

Nombre	**Fecha de lanzamiento**	**Lenguajes soportados por defecto**	**Cuota mercado** [13] (2018)
AWS Lambda [14]	13/11/2014	Java, Python, PowerShell, Go, Node.js, C# y Ruby	44-50 %
Azure Functions [15]	15/11/2016	Java, Python, PowerShell, C#, JavaScript, F# y TypeScript	25-30 %
Google Cloud Functions (GCF) [16]	09/03/2017	Java, Python, Node.js, Go, .NET, Ruby y PHP	20-24 %

Recalcar, que todas estas herramientas soportan un sistema que es capaz de hacer compatible la ejecución de funciones escritas en cualquier lenguaje de programación que no esté en el servicio. Este sistema es llamado *Custom Runtime* en Lambda y GCF, y *Custom Handlers* en Azure [17].

Un *runtime* se refiere a un programa (dentro de los servidores de cada herramienta) que es capaz de traducir el código presente en el manejador de la función que estemos ejecutando a su entorno de ejecución (maneja los procesos que se necesitarán para ejecutar la función) asociado. Será capaz también de lidiar con los eventos que llegan, el contexto con el que se ejecuta y las respuestas entre la herramienta y la función [18].

3.1.1 Funcionalidades básicas

En esta subsección se verán las características principales que nos ofrece cada herramienta remota en cuanto a funcionamiento, accesibilidad, seguridad...

AWS Lambda

AWS Lambda es una herramienta de *Serverless Computing* completa, ya que engloba en ella la creación de funciones *FaaS* y de *backend* escalable *BaaS* (llamado por ellos "aplicaciones"). Presenta las siguientes características destacables [14]:

- Compatibilidad con todos los servicios en la nube de la compañía (*Amazon EC2, Amazon S3*,etc.).
- Los servidores en los cuales se ejecutan las funciones lanzan *Amazon Linux* y la arquitectura de procesador es *arm64* (procesador *AWS Graviton 2*). Está arquitectura, es modular en cuanto a *hardware*, ya que el chip está formado por núcleos especializados en diferentes tareas como lanzamiento de simulaciones o codificación de vídeo.
- Se puede invocar una función que esté a la espera de un evento HTTPS a través de cualquier cliente (navegador web, *curl*). Habrá que añadirle un URL a la función.
- La herramienta ofrece una forma de dividir la concurrencia de funciones (si tenemos varias) en ejecución, para que una función que es llamada de más no nos quite la posibilidad de que otras funciones, que pueden ser también importantes, se retrasen en su ejecución.
- AWS nos da la posibilidad de montar una nube privada entorno a las diferentes funciones que creemos. Estas funciones podrán acceder a bases de datos o a otras funciones que se sirvan entre ellas.
- El límite de escalado no depende de la región donde alojemos la función y se mide en ráfagas de tráfico, es decir, en instancias por minuto. Eso sí, el límite de concurrencia de instancias si dependerá de la región donde se lance la función.
- La herramienta nos da la posibilidad de que a través de contenedores (con su sistema operativo propio y sus extensiones de Lambda) o de ficheros comprimidos con extensión *.zip* podamos lanzar paquetes de código de funciones y no tener que implementarlas una a una desde la utilidad gráfica. Será necesario del uso de la línea de comandos de la compañía.
- En cuanto al uso de líneas de comandos, podremos usar una en el navegador con editor propio o la *AWS CLI* que es una herramienta generalizada instalable que nos sirve para configurar cualquier tipo de infraestructura de AWS. Las consolas nos sirven para listar funciones, editarlas, lanzar paquetes completos (anterior punto), firmar código para asegurar su integridad o crear capas para separar las funciones y sus dependencias (uso de librerías sin necesidad de su inclusión).
- Para garantizar la seguridad en cuanto a los datos, la herramienta nos ofrece métodos de encriptado con TLSv1.2 (uso de HTTPS para conexiones) y además, métodos propios de AWS.
- Para el acceso del administrador, se usa un sistema de gestión de acceso e identidades (IAM) personalizado por la compañía.

Azure Functions

Azure Functions es una herramienta de *Serverless Computing* únicamente de tipo *FaaS* por lo que nos permite escribir funciones en diversos lenguajes de programación sin necesidad de preocuparnos de la gestión y el mantenimiento de el lugar donde se ejecutan. Este sistema tiene estas características que lo diferencian en parte de los diferentes competidores [15]:

- Compatibilidad con todos los servicios en la nube de la compañía (*Azure Cosmos DB, Azure Kubernetes Service*, etc.).
- La forma en la que la herramienta escala la ejecución de funciones, los recursos disponibles para cada función y si tenemos o no nube privada de Azure, depende del plan de pago elegido (hay 3 básicos, *Consumption plan, Premium plan y Dedicated plan*). En los planes de pago más sencillos, la herramienta bonifica a los usuarios que codifiquen su código con el IDE (editor) de Microsoft: Visual Studio. Este *bonus* se aplica en cuanto al número máximo de instancias lanzadas concurrentes en la plataforma.

- Hay que tener en cuenta que las versiones más nuevas de la herramienta son las que soportan los lenguajes de programación anteriormente descritos, en las versiones adecuadas (de los lenguajes) con su *runtime* (con retro-compatibilidad), respecto a Azure Functions.
- Para lanzar proyectos con funciones ya creadas, ofrece un amplio abanico de opciones, por ejemplo: un paquete externo recogido de una *URL* específica, un fichero comprimido con formato *.zip*, un contenedor preparado de *Docker* (sólo para Linux), un repositorio local de *Git*, mediante el protocolo FTP seguro...
- Implementa también el servicio *Azure Functions Proxies* que puede desviar la ejecución de una parte de una función hacia otro recurso disponible, que por ejemplo sea menos costoso o esté menos congestionado.
- Permite el uso de *handlers* (manejadores) personalizados (por ejemplo, ejecutables de Windows) fuera de la función (servidores web ligeros en paralelo). Su uso principal es de permitir partes de la función en lenguajes o tiempos de ejecución no soportados por defecto. Este termino es el usado por Azure Functions para llamar a los *Custom Runtime* que se explicaron tras la Tabla 3.1.
- Para la seguridad, implementa conexiones híbridas (desarrolladas por el proveedor) que consisten en una conexión mediante HTTPS normal pero sólo se permite una conexión por host, por tanto sólo habrá una forma de acceder a la función hasta que está conexión se cierre. Además, las conexiones con herramientas externas de Azure están cifradas. Recalcar también que se protege de los ataques cibernéticos más comunes (*DDoS* y *MITM*) las 24 horas del día.

Google Cloud Functions

Google Cloud Functions, como su nombre indica, es un servicio *Serverless Computing* que sólo implementa *FaaS*. En su nueva versión es capaz de dar un mejor control de escalabilidad y rendimiento, además de implementar una infraestructura más potente. Las siguientes características diferenciarán a esta herramienta de sus competidores [16]:

- Compatibilidad con todos los servicios en la nube de la compañía (*Google Compute Engine, Google Cloud Storage*, etc.).
- En cuanto a escalabilidad, la herramienta permite fijar un máximo de instancias por función para no congestionar otras funciones y un número de estancias mínimas para evitar los arranques lentos de funciones sin estado.
- Ofrece el lanzamiento de funciones de forma externa desde una máquina local, desde la herramienta de repositorios de Google (*Google Cloud Source Repositories*), desde la consola del navegador web o directamente desde la API. Para todas las opciones, excepto API, el código fuente debe estar empaquetado en formato *.zip*.
- La herramienta dispone de un servicio de *logs* y monitoreo de funciones bastante completo. Los *logs* se ejecutarán desde las propias funciones (ejecutando la API para ellos) y la consola nos permitirá visualizarlos con un comando aludiendo a la función específica después de su ejecución. En cuanto a la monitorización de las diversas métricas disponibles (memoria, tiempos de ejecución, número de ejecuciones...), se podrá hacer desde la consola o con *logs* desde la ejecución de la función.
- Permite el uso de variables de entorno asignables a cada función que servirán para usarlas cuando la función esté lanzada y así determinar algunas partes de su comportamiento. También se pueden usar para limitar con una constante las diversas métricas.
- Compatible con las redes privadas virtuales para la nube de Google, pudiendo así, conectar de forma virtual nuestras diversas funciones. Este tipo de redes se pueden crear desde la consola o usando una herramienta externa como *Terraform*.
- Con respecto a la seguridad, incluye: Google IAM para el acceso de la administración de las funciones, identidad para funciones que permite asignarles diversos permisos, seguridad extra para nubes privadas virtuales (VPC, Virtual Private Cloud) con *VPC Service Controls* y diferentes formas de encriptar datos elegidos directamente por el cliente.

3.2 Keda: una herramienta usada por algunas plataformas instalables *Serverless*

La sección tendrá dos partes, la primera es completamente complementaria a la segunda ya que en la subsección siguiente se verán los conceptos básicos necesarios para entender la herramienta. La segunda tratará de describir las partes más importantes de la herramienta.

3.2.1 Conceptos necesarios para entender Keda

Keda es un sistema que usa conceptos básicos de *Kubernetes* que no toda la comunidad debe saber. Además de para Keda, estos conceptos nos servirán para las diversas herramientas que se expondrán en la siguiente sección. Por ello, se desarrolla esta subsección justo antes de la parte que detalla las funcionalidades del sistema. Los conceptos que se van a necesitar son los siguientes [19]:

- **Contenedor *Kubernetes***: son similares a las máquinas virtuales (VM, Virtual Machine), con la diferencia de que los contenedores tienen un aislamiento relajado de las propiedades que se comparten, del sistema operativo instalado, hacia las aplicaciones que usan el contenedor. En cuanto al sistema de ficheros, uso de CPU, memoria, espacio... son prácticamente idénticos a las VM.
  Su principal ventaja frente a las VM, es que al tenerse poco en cuenta el sistema operativo instalado, son fácilmente portables entre proveedores o implementaciones privadas.
- ***Cluster***: conjunto de máquinas, llamadas nodos por separado, que ejecutan aplicaciones en contenedores de *Kubernetes*. Contiene varios nodos de trabajo y un nodo maestro.
- **Unidad mínima (*Pod*)**: objeto más pequeño y simple en cuanto a computación se refiere. Es capaz de representar hasta a un *cluster*, aunque normalmente se usa para ejecutar un contenedor que sea primario en la aplicación o para añadir funcionalidades extra a la misma.
- ***Deployment***: objetos que controlan la configuración de los *Pods* que serán replicados a demanda. Se hará de la siguiente manera, el controlador de escalado de *Kubernetes* mira la configuración guardada en el *Deployment* para saber como y con que imagen se replicarán los *Pods* que han sido requeridos por algún tipo de evento.
- ***StatefulSet***: objetos que gestionan el despliegue y escalado de un conjunto de *Pods* y además, a diferencia de los *Deployments*, son capaces de garantizar la unicidad (a pesar de crearse a partir de la misma especificación, tendrán ids diferentes) y el orden de dichas replicas creadas.
- ***Kubernetes Jobs***: objetos que son capaces de crear una o más unidades mínimas y continuar reintentando su ejecución hasta que se consigue el éxito de un número de ellos especificado. También conocidos como tareas.
- ***Custom Resources***: extensiones de la API de *Kubernetes* que no están necesariamente por defecto instaladas. Representan modificaciones de la instalación de *Kubernetes*.
- **Recursos modificables definidos (CRD, Custom Resource Definition)**: API de *Kubernetes* que es capaz de definir *Custom Resources* sin necesidad de cambiar la instalación base de *Kubernetes*. Es simple y ayuda a crear *Custom Resources* sin necesidad de programar. Estos CRD son capaces de crear nuevos recursos modificados con un nombre y un esquema que se debe especificar.

3.2.2 Características principales de la herramienta

Keda trabaja como un escalador automático de contenedores. Lanza los contenedores suficientes para tratar el número de eventos recibidos. Es capaz de ser instalado en una agrupación de contenedores ya existentes. Necesita mapear grupos de contenedores con los eventos que va a recibir [20].

Cómo su funcionalidad es la de escalar contenedores dependiendo de eventos recibidos, este sistema se puede relacionar de forma clara a herramientas de *Serverless Computing* privado. Esto se debe a que las diversas aplicaciones de este tipo podrían desarrollar su propia forma de escalar o utilizar formas de escalado ya creadas. Keda puede aportar esto último para proyectos *Serverless Computing* de pequeño presupuesto o directamente comunitarios. Uno de los casos en lo que ocurre esto es OpenFunction.

Es recalcable también que es compatible con Azure Functions y Alibaba Cloud, entre otros del mundo remoto, si se lanzan las funciones en *Kubernetes*.

Es capaz de desempeñar 2 roles en su funcionamiento con *Kubernetes*. Estas formas de trabajar son las siguientes:

- **Agente**: activa y desactiva objetos *Deployment* para el escalado desde o hacia 0 sin eventos. Tras la activación, comienza la espera de eventos que se tengan asignados.

- **Métricas**: actúa como servidor de métricas para *Kubernetes*. Es capaz de bajar o subir el ritmo de escalado de los contenedores asociados mirando los parámetros que se monitorizan (por ejemplo, la longitud de la cola de eventos o el retardo que sufre el escalador). La herramienta tratará de estabilizar las mediciones de las métricas en el punto en el que se hayan configurado.

Tras está pequeña definición introductoria de la herramienta, se explicará por partes su funcionalidad:

- **Escalado**: la herramienta tiene 2 formas principales de escalar.
  - La primera de ellas es el escalado mediante *Deployments* y *StatefulSets*. De esta forma, Keda actúa como monitor de eventos (con los diversos disparadores disponibles) para alimentar con información a los controladores de contenedores para que así actúe el escalador automático de *Kubernetes* (HPA).
  - La otra forma es el escalado de *Kubernetes Jobs* basado en eventos. Esta manera se usa para ejecuciones largas.
- **Disparadores**: el sistema implementa un número bastante extenso de disparadores de eventos, en total 61. Además de tener la capacidad de generar contenedores, también sirven para desactivar o activar los *Deployments* o *StatefulSets*. Algunos ejemplos de disparadores podrían ser: Apache Kafka, AWS CloudWatch, por CPU, por memoria, de PostgreSQL...
- **Autenticación**: frecuentemente los diversos escaladores que utiliza el sistema necesitarán una forma de autenticarse con sus proveedores originales, ya que sin esta autenticación no es posible el acceso a la configuración o a la propia comprobación de eventos. La herramienta nos da patrones seguros para gestionar los flujos de autenticación. Éstos son:
  - Configuración de la autenticación para *ScaledObject* (CRD el cuál es usado para comunicar a Keda cómo debería escalar la aplicación y qué disparadores mirar).
  - Reuso de credenciales o delegación de autenticación con *TriggerAuthentication*. Esta forma permite agregar más parámetros (y más avanzados) de autenticación separados de los *ScaledObjects* y de los contenedores ya lanzados.
  - Reuso de credenciales globales para disparadores de *clusters*.
- ***Logging***: La herramienta nos proporciona un sistema de registros incluidos en el controlador por defecto de *Kubernetes*. Estos mensajes son importantes para comprobar por qué algo no está funcionando correctamente y así poder solucionarlo.

3.3 Plataformas instalables o privadas

La sección en la que nos encontramos, se encarga de comparar y contextualizar las diferentes plataformas privadas que he decidido probar e instalar. Se han elegido estas herramientas porque su desarrollo es supervisado por el CNCF y porque son probablemente las más populares del sector.

El proceso de instalación y de programación de una función de prueba se verá en el próximo capítulo para todas las herramientas mencionadas en este capítulo teórico.

Tabla 3.2 Comparativa de plataformas instalables.

Nombre	**Fecha de lanzamiento (versión estable)**	**Lenguajes soportados por defecto**	**Características principales**
CNCF Knative [21]	02/11/2021 (1.0) nace 18/07/2018	C#, Go, Java (Spark), Java (Spring), Kotlin, Node.js, PHP, Python, Ruby, Scala, Shell	- Divide la herramienta en *serving* y en *eventing*. - Sólo funciona con contenedores de *Kubernetes*. - Objetos CRD que usa: *services*, *route*, *configuration* y *revision*.
Apache OpenWhisk [22]	31/10/2018	.Net, Go, Java, JavaScript, PHP, Python, Ruby, Swift	- Cualquier lenguaje añadible (guía incluida). - Válido para nube privada o pública (mediante proveedor). - En vez de funciones, acciones.
OpenFaaS OpenFaaS [23]	08/08/2017	Go (en diversas formas), Python, Node.js, Java, C#, Ruby y PHP.	- Uso recomendado de *Kubernetes*, motor propio para tareas ligeras (*faasd*). - GUI o CLI. - Alta colaboración de la comunidad (nacida de ella).
QingCloud OpenFunctions [24]	17/05/2021	Go, Node.js, Python, Java	- Para servir funciones necesita de Knative (modo síncrono) o de Keda (modo asíncrono (sólo disponible para Go)). - Infraestructura de eventos variada (disparador, bus y fuente). - HTTP y *Cloud Event* (Go)

La anterior tabla da una primera impresión sobre la cronología en la que fueron lanzadas las diferentes herramientas (versión estable es referido a que el propio desarrollador las considera que están fuera de su fase de desarrollo inicial), además de dar una pequeña visión de funcionalidad mediante los lenguajes soportados por defecto (lenguajes en los que puedes programar funciones, sin necesidad de modificar la herramienta) y algunas características destacables que hacen que se diferencien de las demás presentadas.

3.3.1 Knative

Sistema automatizado que ayuda a los desarrolladores a gestionar y mantener procesos basados en *Kubernetes* lo que hace que éstos se centren más en el desarrollo de aplicaciones que en la pura configuración [21]. Como ya se comentó en la tabla, la herramienta se divide en 2 partes:

- ***Serving:*** la parte que se encarga de servir y lanzar los contenedores a partir de peticiones del protocolo HTTP (también disponible con HTTPS), además, provee componentes que son capaces de proporcionar:
  - Despliegue de forma rápida y automática de contenedores *Serverless*.
  - Escalado automático, incluso de *Pods* (ver Subsección 3.2.1) desde cero.
  - *Snapshots* (imágenes en un punto temporal) de configuraciones y código desplegado.

  Esta parte define un conjunto de objetos con la forma de CRD (ver Subsección 3.2.1) para que se puedan instalar fácilmente en la API de *Kubernetes* que se esté utilizando. Estos objetos especifican cómo el flujo de trabajo *serverless* se comporta en el conjunto de contenedores:
  - *Service*: objeto que representa una instanciación de un único entorno de contenedores *serverless*. Incluye, dirección de red con la cuál el servicio es alcanzable como para lanzar el código asociado y ejecutar su configuración.
  - *Route*: objeto que representa el estado actual de la petición HTTP comparado con el conjunto de objetos *revision* (abajo explicado). Ésto permite el lanzamiento progresivo de aplicaciones sin servidor, ya que un objeto de tipo ruta soporta peticiones independientes de los diferentes códigos de aplicación y configuraciones. Estos objetos son propiedad de un *service* y su control será desarrollado desde éste.
  - *Configuration*: objeto que representa el estado futuro deseado de una representación única de una aplicación y su configuración en un único contenedor (después de que el despliegue haya sido exitoso). Este objeto nos da una plantilla para la creación de objetos *revision* como futuro estado de los cambios de la aplicación. Estos objetos, si están asociados a un objeto *service*, no deben ser manipulados directamente por el desarrollador, sino que deben ser cambiados mediante el controlador del servicio.
  - *Revision*: objeto que representa una instantánea (sin estados) del escalado automático en un determinado tiempo de la ejecución del código de la aplicación y su configuración. Permite lanzar y retirar cambios de la aplicación, cambiando el rutado HTTP entre los nombres de servicio y las instancias de revisión. Los desarrolladores no deben ser capaces de crear objetos de este tipo directamente, ya que éstos se deben crear en respuesta de actualizaciones de objetos *configuration*.
- ***Eventing:*** la parte que se encarga de servir herramientas para catalogar eventos desde productores de los mismos hasta sumideros, permitiendo así a los desarrolladores usar una arquitectura basada en eventos con sus aplicaciones. Knative usa el estándar de peticiones HTTP POST para enviar y recibir eventos entre productores y sumideros.
  Para describir esta parte, de una forma más completa, hace falta revisar las componentes de este proceso que definen diferentes formas de procesar la llegada de eventos. Las cuáles son:
  - Fuentes (*Sources*): principales productores de eventos, éstos pueden ser enviados hacia un sumidero o directamente hacia un suscriptor.
  - Intermediarios y Disparadores (*Brokers and Triggers*): facilitan un modelo de eventos basado en que los *brokers* son sumideros y filtros de eventos. Éstos son capaces de distribuir de forma uniforme (dependiendo de los atributos del evento) a los consumidores de los mismos usando diferentes *triggers*. Los disparadores se asocian a un suscriptor o a varios (con el mismo disparador). Los suscriptores ejecutarán la función.
  - Canales y Suscripciones (*Channels and Subscriptions*): facilitan un modelo de eventos basado en que los canales pueden recibir varios eventos de diferentes fuentes y el canal es capaz de redistribuir todos los eventos recibidos hacia sus diferentes suscriptores haciendo que diferentes sumideros reciban lo mismo simplemente conectándose a este "bus".
  - Registro de eventos (*Event registry*): conserva una especie de catálogo o lista de los tipos de eventos que cada *broker* es capaz de consumir.

3.3.2 OpenWhisk

La herramienta de la organización Apache sigue un modelo de desarrollo claro y conciso (sigue de forma evidente el modelo presente por defecto en herramientas *FaaS*). Siempre será igual, lo cuál simplifica el

trabajo para el programador, y además es ciertamente parecido al modelo que se usa en las plataformas remotas por los diferentes proveedores [22]. En la Figura 3.1 se ve un esquema de este modelo:

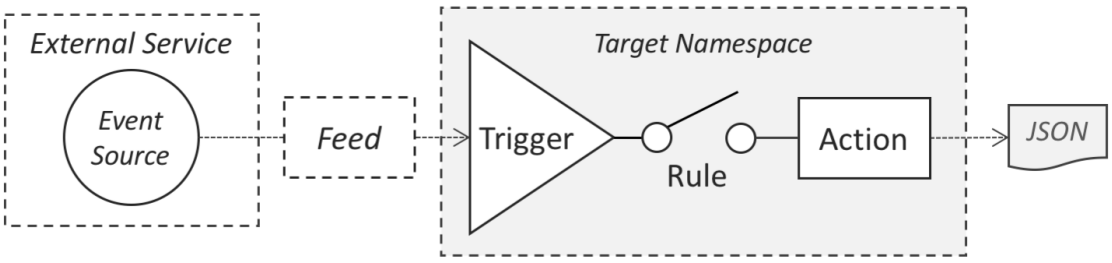

Figura 3.1 Modelo de programación de OpenWhisk extraído de [22].

A continuación, voy a detallar las partes más importantes del modelo, señalando su funcionalidad y sus componentes:

- **Alimentación (*Feed*):** flujo de eventos que tienen como destino el mismo disparador. La herramienta soporta una API abierta para que cualquier usuario pueda exponer cualquier productor de eventos como *feed* en un paquete. La arquitectura para ello se compone de al menos 3 patrones para crear esta parte del modelo a nuestro gusto. Esos 3 patrones son:
  - Ganchos (*Hooks*): en este patrón, se activa el *feed* mediante un servicio *webhook* HTTP (retrollamada HTTP tras realizar un HTTP POST). Este patrón nos permite asociar directamente un disparador a una URL a la que hagamos POST. Ésta es una forma sencilla de implementar *feeds* que no se activen con frecuencia.
  - Votaciones (*Polling*): en este patrón, se ordenará a la *action* de OpenWhisk que sondee de forma periódica la fuente para recabar los datos necesarios. Cuando se tengan los datos se activará el disparador. Fácil de construir, pero la frecuencia de eventos está limitada a la frecuencia de sondeo.
  - Conexiones (*Connections*): en este patrón, se lanzará un servicio en algún lugar que mantenga una conexión continua para alimentar una fuente. Esta implementación se puede realizar mediante *long polling* (conexión sin esperar respuesta, el que responde lo puede hacer cuando quiera ya que el que pide deja la conexión abierta) o mediante el envío de notificaciones.
- **Disparador y Regla (*Trigger and Rule*):** Esta parte del modelo hace que el sistema sea capaz de manejar eventos ya sean internos o externos, debido a que para que se lance una acción debe activarse un disparador. Los disparadores se activarán a partir de un diccionario (pares clave-valor en JSON). Este diccionario puede ser producido por el evento al que está asociado (o por alguien para probar la función). Cada lanzamiento de un disparador tendrá asociado un id propio. Las reglas servirán, cómo la imagen indica, para ser capaces de lanzar una acción asociándose con los disparadores. También es posible que un disparador lance múltiples reglas.
- **Acción (*Action*):** las acciones de OpenWhisk son funciones sin estado que se lanzan en su plataforma. Las acciones pueden ser creadas mediante 3 formas: usando los lenguajes soportados (se pueden incluir más mediante una implementación propia o ya existente o con el *Action Loop Engine* que da un patrón para implementarlo), mediante un ejecutable binario o directamente usando contenedores ya configurados.
  La herramienta incluye una utilidad de línea de comandos que hace más sencilla la configuración e invocación de estas funciones.

3.3.3 OpenFaaS

OpenFaaS es una herramienta desarrollada por la comunidad para hacer más fácil (codificación poco repetitiva o incluso con GUI) a desarrolladores el lanzamiento en contenedores de *Kubernetes* de funciones y microservicios basados en eventos.

Cómo su nombre indica, su forma de actuar y lanzar funciones está directamente basada en la forma por defecto de *FaaS*. Ya entrando más en cómo funciona el sistema, se describirán aquí las funcionalidades básicas que lo componen y algo breve sobre la configuración del mismo [23]:

- La aplicación permite lanzar los contenedores que crea en una plataforma remota o en local. En remoto lo hará en las plataformas de *Kubernetes* de los proveedores más famosos. Éstas son: *Amazon EKS*, *Azure AKS*, *Digital Ocean Kubernetes* y *Google Kubernetes Engine*.
- La herramienta basa su configuración en ficheros *YAML*, además tiene un fichero para configuración general llamado *stack*.
- El sistema es capaz de trabajar con funciones síncronas y asíncronas. Si se usan las síncronas, la conexión esperará a la finalización de la misma para recibir la respuesta. Si se usan las asíncronas, más indicadas para códigos pesados (mucho tiempo de ejecución), la función implementará un *callback* para que, cuando se haya acabado la ejecución, se avise al que pidió la ejecución y se le envíe la información. Las conexiones y *callbacks* se realizarán mediante HTTP.
- En cuanto al procesamiento de eventos y el lanzamiento de disparadores, la herramienta es compatible con una larga lista (la mayoría usan HTTP para comunicar su información) que expongo aquí: *Cron, HTTP / webhooks, Async / NATS Streaming, using faas CLI* y mediante el patrón conector de eventos *Apache Kafka, AWS SQS, Cron Connector, MQTT Connector, NATS Pub/sub, Minio / S3, AWS SNS, CloudEvents, RabbitMQ, IFTTT, VMware vCenter, etc.*.
- Sobre el escalado automático implementado, funciona horizontalmente con funciones entre un mínimo y un máximo número de réplicas o incluso desde cero. Se configuran por función. Se implementan 3 tipos de escalado:
  - *RPS*: basado en peticiones por segundo completadas, bueno para funciones de ejecución rápida.
  - *Capacity*: basado en peticiones que se están cursando, bueno para funciones que tardan en ejecutarse o no tienen asignado un gran número de peticiones al mismo tiempo.
  - CPU: basado en el uso de CPU asignado, se usa cuando los otros métodos no están funcionando bien.
- El sistema implementa un *WatchDog* que es capaz de monitorizar la ejecución e inicializar funciones, además gracias a esta herramienta se podrá ejecutar cualquier binario como función.

3.3.4 OpenFunction

OpenFunction es una herramienta de código abierto de tipo *FaaS* con el objetivo clásico de las plataformas de *Serverless Computing*, el de centrar al desarrollador en la escritura de código en vez de en la configuración del escenario.

El sistema tiene una serie de conceptos y componentes que explicaré de forma breve a continuación [24]:

- **Funciones:** las funciones en esta implementación son CRD (ver Subsección 3.2.1) (como en *Knative*) que se pueden definir y gestionar. Actúan como una descripción de la aplicación, desde el código fuente usado que se construirá hasta como se desarrollará el flujo de trabajo a lo largo de la ejecución.
- **Constructor:** el *builder* es otro objeto de tipo CRD que define el trabajo de construcción para generar imágenes de aplicaciones desde el código fuente de las funciones. Esta implementación usa *Shipwright* (infraestfructura de construcción de contenedores *Kubernetes*) y *Cloud Native Buildpacks* (tranformador de código en imágenes de contenedores).
- **Parte de *Serving*:** esta parte aspira a servir un objeto CRD mediante la ejecución de funciones de manera elástica a través de un escalado dinámico. Para la mayoría de implementaciones de código la aplicación sólo funciona con *Knative Serving* (HTTP), pero para Go implementa una herramienta propia: *OpenFuncAsync* (usa *Keda* y *Dapr* y puede recibir diferentes tipos de eventos).
- **Eventos:** en cuanto a esta parte, la plataforma los desarrolla 3 partes: implementará una fuente de eventos (compatible con *Kafka*, *Cron* y *Redis*) que servirá para lanzar funciones síncronas de forma directa (enviarán también eventos al bus), contiene un bus de eventos que es responsable de agregar todos los eventos recibidos para hacerlos persistentes mediante un *broker* (normalmente una cola de mensajes) y por último los disparadores que avisarán a la función asíncrona que tienen conectada de la llegada de un evento al bus.
- **Dominio:** define una entrada unificada para las funciones síncronas, por ejemplo una URL. Se usa para definir un dominio por *cluster*, que además necesita un controlador para ingresar.

4 Instalación y configuración de varias herramientas *Serverless Computing* para uso privado

La aerodinámica es para fracasados que no saben hacer motores

Enzo Ferrari

En este capítulo se verán los diferentes procesos de instalación y configuración de los servicios de *Serverless Computing* privado. Tras este capítulo, el desarrollador que lea el Capítulo 5 debe ser capaz de crear funciones sin problemas.

Es necesario aclarar que este capítulo está enfocado para ser leído por un administrador de sistemas. Esto es debido a que en él se comentan los procesos de instalación de las herramientas y además algunos procesos de configuración breves.

4.1 Entorno de trabajo

Para la instalación de las diversas plataformas se usará mi ordenador de sobremesa al que le he instalado Ubuntu 22.04 LTS. Este sistema operativo será necesario para mejorar el rendimiento de los diversos contenedores que soportarán las herramientas y unificar los procesos de instalación.

El ordenador de sobremesa utilizado consta de un procesador Intel Core i7 4770 (4 núcleos físicos, 8 hilos virtuales) y de 12 GB de RAM, así que, según las especificaciones técnicas de las herramientas con las que se va a trabajar, éstas funcionan sin ningún tipo de problema.

4.2 Instalación de Knative del CNCF

En está sección se verán los diferentes pasos en la instalación, destacando los requisitos necesarios para la misma y se realizará una configuración básica. Si se quieren recordar los conceptos teóricos básicos visitar la Subsección 3.3.1.

En particular se hará la instalación de Knative denominada *quickstart* que es la recomendada para la experimentación de desarrolladores y que además consume menos recursos *hardware* ya que, mi ordenador de sobremesa, en el que estoy realizando el TFG, no soporta en potencia (según Knative) las versiones instaladas mediante ficheros YAML o el Knative Operator [21].

De hecho, soportará por poco la versión *quickstart*, que requiere de 3 núcleos y 3 GB de RAM.

4.2.1 Requisitos previos

Para que la herramienta funcione correctamente, será necesario instalar diversas aplicaciones, las cuales son:

- Una herramienta para crear *clusters* de *Kubernetes* dentro de un contenedor local de Docker: *kind*.
- La herramienta de línea de comandos de *Kubernetes*, *kubectl*.

- La herramienta de línea de comandos de Knative, *kn*.

En esta subsección se verá cómo se instalan cada una de las dependencias que se han presentado previamente. Probablemente, se hará referencia a esta subsección en partes de requisitos previos de las otras herramientas.

Instalación de *kind*

Para conseguir la instalación de esta herramienta es necesario primero instalar en Ubuntu la herramienta de Docker, Docker Desktop. En un principio se iba a usar Windows con WSL 2, pero Docker Desktop no es compatible con OpenFunction y para unificar se hará todo en el sistema Linux que permite instalar Docker Engine por separado y conjuntamente con la herramienta de escritorio.

En el Código 4.1 se visualiza como se instala Docker Desktop en Ubuntu. También habrá que gestionar el inicio de sesión en Docker Hub desde el programa de escritorio, para ello se sigue a rajatabla el tutorial de la documentación de Docker presente en *https://docs.docker.com/desktop/get-started/* (parte de Ubuntu).

Código 4.1 install_docker.sh.

```
# Instalación del repositorio de Docker
sudo apt-get update
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
     /etc/apt/keyrings/docker.gpg
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
     https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /
    dev/null

# Descarga del paquete .deb
wget https://desktop.docker.com/linux/main/amd64/docker-desktop-4.11.0-amd64.
    deb

# Instalación con el paquete .deb (el repositorio de docker es necesario porque
     tiene dependencias a instalar aparte)
sudo apt-get update
sudo apt-get install ./docker-desktop-4.11.0-amd64.deb
```

Tras este paso previo ya podemos proceder a la instalación de *kind*. Para Ubuntu, *kind* no está disponible con un instalador de paquetes, por tanto, se procederá mediante la descarga directa del ejecutable binario. En el Código 4.2 se pueden ver los pasos necesarios para completar la misma [25].

Código 4.2 install_kind.sh.

```
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.14.0/kind-linux-amd64
chmod 744 ./kind
sudo mv ./kind /usr/local/bin/kind
```

Instalación de *kubectl*

Para poder usar *kind* y Knative en un futuro es necesario instalar la utilidad de línea de comandos de *Kubernetes*, esta nos permitirá lanzar aplicaciones, revisar y gestionar recursos de los *clusters* y ver registros. En el Código 4.3 se pueden apreciar los pasos necesarios seguidos para completar el proceso de instalación [19].

Código 4.3 install_kubectl.sh.

```
# Necesidad de instalar la dependencia apt-transport-https
sudo apt-get install -y apt-transport-https
```

```
# Hay que instalar el repositorio de Kubernetes para Ubuntu
# Clave publica necesaria para añadir el repositorio
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://
    packages.cloud.google.com/apt/doc/apt-key.gpg
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https
    ://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.
    list.d/kubernetes.list

# Ya se puede proceder a la instalación por paquetes
sudo apt-get update
sudo apt-get install -y kubectl

# Movemos al directorio donde tenemos todas las instalaciones
sudo mv /usr/bin/kubectl /usr/local/bin/
```

Instalación de *kn*

La instalación de esta línea de comandos es necesaria porque nos servirá para crear recursos, tales como servicios de Knative y fuentes de eventos, sin necesidad de crear o modificar el fichero de configuración YAML de manera directa. También, ayudará a completar procesos complejos en cuanto al auto-escalado. En el Código 4.4 se explican los diversos pasos necesarios para completar el proceso de instalación. En este caso, vuelve a ocurrir que no tenemos instalación mediante paquetes [21].

Código 4.4 install_kn.sh.

```
# Descargamos el binario de kn y lo procesamos
wget https://github.com/knative/client/releases/download/knative-v1.5.0/kn-
    linux-amd64
mv kn-linux-amd64 kn
chmod 744 kn

# Movemos el ejecutable al lugar donde tengamos las aplicaciones
sudo mv kn /usr/local/bin
```

4.2.2 Instalación

En este caso, tras instalar todos los requisitos previos, sólo nos queda lanzar el plugin para la versión *quickstart* de Knative. Será necesario el uso del binario directamente, ya que mediante paquetes sólo está disponible para MacOS. También se podría instalar con el compilador de *GO*, pero en mi caso, es preferible no cargar más la instalación de Ubuntu para mejorar el rendimiento. En el Código 4.5 se pueden visualizar los pasos a seguir para finalizar la instalación [21].

Código 4.5 install_Knative.sh.

```
# Descarga y procesado del plugin de quickstart
wget https://github.com/knative-sandbox/kn-plugin-quickstart/releases/download/
    knative-v1.5.1/kn-quickstart-linux-amd64
chmod 744 kn-quickstart-linux-amd64
mv kn-quickstart-linux-amd64 kn-quickstart

# Movemos al lugar de las aplicaciones
sudo mv kn-quickstart /usr/local/bin/
# Para comprobar que el plugin está correcto lanzar: kn quickstart --help

# Lanzamos el plugin con kind
# Hay que tener iniciado Docker previamente para que se pueda crear el cluster
```

```
kn quickstart kind
# Para comprobar la correcta instalación: kind get clusters
# Debe contener un cluster llamado Knative
```

Para evitar gasto de *hardware* cuando no se esté usando Knative, se puede parar el contenedor que lo gestiona desde la pestaña *Containers* de la GUI de Docker Desktop. También, en sí, tener el motor de Docker encendido consume rendimiento, para cerrarlo basta con cerrar la GUI y terminarlo desde la barra de herramientas o administrador de tareas, como si de un proceso en segundo plano se tratase.

4.3 Instalación de OpenWhisk de Apache

En está sección se verán los diferentes pasos en la instalación, destacando los requisitos necesarios para la misma y finalmente, se realizará una configuración básica. Si se quieren recordar los conceptos teóricos básicos visitar la Subsección 3.3.2.

OpenWhisk puede ser usado tanto en nube local como en nube remota, en este caso se instalará la versión pensando en uso local. Es necesario aclarar esto ya que, la instalación cambia sustancialmente porque se deberían realizar varios pasos extra si se hace para nube remota [22].

4.3.1 Requisitos previos

Para que la herramienta funcione correctamente, únicamente necesita de un sistema que sea capaz de lanzar contenedores, la organización recomienda usar contenedores de *Kubernetes*, por tanto con la instalación de Docker Desktop y *kind* que se realizó y explicó previamente en la Subsección 4.2.1 es suficiente. Se podrían instalar otras herramientas como *Minikube*, pero ya teniendo instalada una como *kind*, no es necesario instalar otra de este tipo.

También, será obligatorio para realizar el proceso, la instalación del desplegador de paquetes de *Kubernetes*, es decir, *helm*. Esta herramienta sirve para instalar paquetes remotos de varios contenedores que cumplirán una función de aplicación, en este caso los contenedores necesarios para que OpenWhisk funcione en un *cluster* local. En el Código 4.6 se explican los pasos a realizar para conseguir esta herramienta.

Código 4.6 install_helm.sh.

```
# Descargamos el comprimido con el ejecutable y lo extraemos en la carpeta de
    trabajo
wget https://get.helm.sh/helm-v3.9.0-linux-amd64.tar.gz
tar -xvf helm-v3.9.0-linux-amd64.tar.gz

# Movemos al lugar de instalación de las aplicaciones
sudo mv linux-amd64/helm /usr/local/bin/
```

4.3.2 Instalación

Para que la herramienta sea instalada correctamente y que se tenga una experiencia adecuada en el uso de ésta, al menos, hay que realizar 3 pasos. Estos son:

1. La instalación de la línea de comandos de OpenWhisk (*wsk*).
2. La instalación del lanzador de acciones de OpenWhisk (*wskdeploy*).
3. Un cluster de *Kubernetes* adaptado para OpenWhisk.

Instalación de *wsk*

Wsk es el sistema CLI (Command Line Interface) de la herramienta y su función principal es la de facilitar la creación, ejecución y configuración de las entidades que presenta OpenWhisk (invocar acciones, configurar disparadores y reglas, etc.).

La instalación se realizará mediante la descarga del ejecutable, ya que hay una instalación con paquetes pero es con un instalador no por defecto (no viene directamente en Ubuntu y para no instalar herramientas en exceso se descarga únicamente el ejecutable). En el Código 4.7 se visualizan los pasos seguidos para completar el proceso de instalación.

Código 4.7 install_wsk.sh.

```
# Descargamos el comprimido con el ejecutable y lo extraemos en la carpeta de
    trabajo
wget https://github.com/apache/openwhisk-cli/releases/download/1.2.0/OpenWhisk_
    CLI-1.2.0-linux-amd64.tgz
tar -xvf OpenWhisk_CLI-1.2.0-linux-amd64.tgz

# Movemos al lugar de instalación de las aplicaciones
sudo mv wsk /usr/local/bin/
```

Instalación de *wskdeploy*

Wskdeploy es el sistema que ayuda a lanzar y gestionar una infraestructura completa de OpenWhisk (*Packages, Actions, Triggers, Rules* y *API*) a través de ficheros YAML llamados *OpenWhisk Manifest Files*. En este caso sólo se puede instalar mediante la descarga del ejecutable.

Hay que tener en cuenta que se ha usado finalmente el sistema CLI. Esto se debe a que éste sirve para crear acciones, pero también para comprobar logs y configurar parámetros de la herramienta. En cambio, *wskdeploy* únicamente sirve para crear acciones y otros objetos de OpenWhisk a partir de ficheros YAML. Finalmente, se ha usado para hacer una comprobación rápida de que OpenWhisk funciona con el ejemplo de Python de la documentación.

En el Código 4.8 se presentan los pasos a seguir para instalar esta herramienta.

Código 4.8 install_wskdeploy.sh.

```
# Descargamos el comprimido con el ejecutable y lo extraemos en la carpeta de
    trabajo
wget https://github.com/apache/openwhisk-wskdeploy/releases/download/1.2.0/
    openwhisk_wskdeploy-1.2.0-linux-amd64.tgz
tar -xvf openwhisk_wskdeploy-1.2.0-linux-amd64.tgz

# Movemos al lugar de instalación de las aplicaciones
sudo mv wskdeploy /usr/local/bin/
```

Creación de un cluster de *Kubernetes* adaptado para OpenWhisk

Tras instalar las herramientas principales, OpenWhisk, al ser instalado en Docker Desktop, necesita de un contenedor de Docker especializado con alguna configuración diferente. Éste se creará con *kind* y para crearlo se necesita de un archivo YAML proporcionado por el desarrollador en su documentación. El archivo es el contenido en el Código 4.9. Sin la declaración de *extraPortMappings*, no será posible acceder al interior del contenedor, por tanto las peticiones a la API Host no serían posibles y las acciones no funcionarían.

Código 4.9 kind-cluster.yaml.

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  extraPortMappings:
    - hostPort: 31001
      containerPort: 31001
```

Con el archivo anterior, para conseguir lanzar el contenedor con *kind* es necesario ejecutar el Código 4.10 que se presenta a continuación.

Código 4.10 Creación del contenedor.

```
# Creación del contenedor con kind y Kubernetes dentro
kind create cluster --config kind-cluster.yaml --image kindest/node:v1.23.3 --
    name openwhisk-kcluster

# Creamos el namespace para separar visualmente y conceptualmente (comprobar
    pods y deployments más fácilmente) con los contenedores propios de
    Kubernetes
kubectl create namespace openwhisk

# Asignamos el rol de invocador a todos los contenedores para que tengan
    potestad de crear otros
kubectl label nodes --all openwhisk-role=invoker
```

Tras esto habrá que desplegar OpenWhisk con *helm* y con una configuración de ejemplo YAML (no es la misma que ofrece el desarrollador) representada en el Código 4.11.

Código 4.11 mycluster.yaml.

```
whisk:
  ingress:
    type: NodePort
    apiHostName: 172.18.0.2
    apiHostPort: 31001

nginx:
  httpsNodePort: 31001
```

Para instalar con *helm* habrá que seguir el Código 4.12 que se presenta a continuación. Se usará el instalador presente en el GitHub de OpenWhisk.

Código 4.12 Comandos para lanzar la instalación en el cluster.

```
# Descargamos el repositorio y cambiamos al directorio
git clone https://github.com/apache/openwhisk-deploy-kube.git
cd openwhisk-deploy-kube

# Lanzamiento de la instalación
helm install owdev ./helm/openwhisk -n openwhisk -f ../mycluster.yaml

# Si da error probar la desinstalación con los siguientes comandos y volver a
    instalar:
helm uninstall owdev -n openwhisk
kubectl -n openwhisk delete pod --all

# Para comprobar como va el proceso de instalación de los pods necesarios:
kubectl get pods -n openwhisk --watch
# Hasta que no esté el Pod que contiene en su nombre install-packages en
    Completed no se puede dar por terminada la instalación
```

4.3.3 Configuración

En esta subsección se verán las acciones que son necesarias para que se pueda crear una implementación básica en Python en esta herramienta.

Es necesario aclarar que estas configuraciones realizadas son para el CLI de OpenWhisk. Éste necesita saber cuál es la dirección IP y puerto que se están usando para su API. Para recoger información de la API se usa HTTP y autenticación HTTP, por ello es necesario también configurar un usuario para que el CLI sea

capaz de acceder a la información. Se usa el usuario por defecto o usuario invitado para no tener que crear un usuario nuevo para la autenticación HTTP.

En el Código 4.13 se ven los comandos básicos para ser capaz de poner en funcionamiento la utilidad del tutorial (un hello-world típico pero para Java Script). Es decir, la configuración necesaria para hacer funcionar alguna acción en la herramienta de la forma más básica posible.

Código 4.13 Comandos de configuración básica.

```
# Ver los contextos para seleccionar
kubectl config get-contexts

# Seleccionar el cluster de OpenWhisk para ver el estado de sus contenedores
    internos
kind get clusters # Listar los creados por kind
kubectl config use-context kind-openwhisk-kcluster

# Ponemos la ruta del fichero de configuración (se guardan los parámetros
    introducidos por la CLI)
export WSK\_CONFIG\_FILE=$HOME/.wskprops

# Creamos el fichero de configuración para evitar errores
touch $HOME/.wskprops

# Localización de la API
wsk property set --apihost localhost:31001

# Recoger clave de autorización de guest para asignarla a el CLI
kubectl -n openwhisk -ti exec owdev-wskadmin -- wskadmin user list guest

# Asignación de la clave de guest recogida
wsk property set --auth <clave_recogida>

# Comprobar que hay conectividad con la Api Host (no debe devolver un error en
    la petición HTTPS). La opción -i será necesaria (en todos los casos de uso
    del CLI) ya que las peticiones son HTTPS y sin ella nos daría un error con
    el certificado usado.
wsk -i list -v
```

Al tener una librería externa que utilizar, habrá que modificar el *runtime* de Python que tiene por defecto OpenWhisk, ya que esta librería no viene instalada. Para conseguir esto, se ha seguido la guía de uno de los desarrolladores de OpenWhisk [26].

En esta guía se explica que para conseguir instalar librerías externas de Python en OpenWhisk hay que crear entornos virtuales de Python. Estos entornos son estructuras de ficheros que incluyen lo necesario para que Python pueda ser ejecutado desde esta estructura sin afectar al Python ya instalado en el ordenador. En estos entornos se pueden instalar librerías externas para su prueba sin que éstas sean instaladas en el ordenador donde se ejecuta el entorno. OpenWhisk añade el entorno virtual completo dentro de la acción para así poder ejecutar otro Python en vez del que tiene por defecto. Así, por tanto, se pueden añadir librerías al *runtime* de la herramienta.

Aunque estos entornos se añadan por completo, deben ser ejecutados por la versión de Python con las que se crearon para que sean compatibles. Por ello, posteriormente se hará la instalación de un Python en la versión que incluye OpenWhisk en su *runtime*.

En un principio se iba a usar la librería de procesamiento de imágenes para Python OpenCV como en las otras herramientas, pero esta librería es muy completa y pesada; y OpenWhisk tiene un límite para el tamaño máximo para una acción, 48MB. OpenCV, con numpy (una de sus dependencias), supera de sobra este número (el entorno virtual se queda en unos 80MB).

Por ende, se va a usar Pillow, una librería de procesamiento de imágenes para Python que no incluye tantas funcionalidades pero que sirve para el propósito aquí necesitado. Al no ser tan completa es menos pesada y cabe perfectamente en una acción de OpenWhisk.

Debido a que el *runtime* utilizado es el por defecto de OpenWhisk y éste usa la versión 3.7.11 de Python (ha sido necesario comprobarlo, ya que no aparece en la documentación, con una acción que devolvía un mensaje con la versión de Python), será importante crear el entorno virtual de Python con esta versión como ya se adelantó antes. Esto es debido a que si no se realiza de esta manera podrían aparecer errores de incompatibilidad con las librerías instaladas.

Para ello se ha realizado la instalación de esta versión de Python mediante el código fuente de la misma. En el Código 4.14 se pueden apreciar los pasos necesarios para conseguir la instalación.

Código 4.14 install_python3_7_11.sh.

```
# Descargamos dependencias
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-
    dev libc6-dev libbz2-dev
sudo apt update
sudo apt upgrade
sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss
    3-dev libssl-dev libsqlite3-dev libreadline-dev libffi-dev wget libbz2-dev
    -y

# Descargamos, descomprimimos y nos movemos a la carpeta con el código fuente
wget https://www.python.org/ftp/python/3.7.11/Python-3.7.11.tgz
tar xfv Python-3.7.11.tgz
cd Python-3.7.11/

# Para conseguir la instalación ejecutar de uno en uno los siguientes comandos
# --enable-optimizations hace que en la compilación se realicen diversos test
    que comprobarán el funcionamiento, esto hará que la instalación tarde más
    en terminar
./configure --enable-optimizations
# Make con -j se hará más rápido
make -j <num_hilos_procesador>
# Se hace con altinstall para no sobreescribir el python ya instalado en el
    sistema
sudo make altinstall


# Instalación mediante paquetes de "deadsnake" (se ha usado la mediante código
    fuente)
# Prerrequisitos
sudo apt update
sudo apt upgrade
sudo apt install software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa

sudo apt install python3.7

# Instalación de pip
python3.7 -m ensurepip --upgrade
```

En el Código 4.15 se presentan los comandos necesarios para acometer la creación del entorno virtual y la instalación de las dependencias que se necesitarán para ejecutar la acción.

Código 4.15 Comandos para añadir librerías al Runtime de Python de OpenWhisk.

```
# Instalación de la herramienta para creación de entornos virtuales avanzados
python3.7 -m pip install virtualenv

# Creación del entorno virtual
mkdir envVir
cd envVir/
python3.7 -m virtualenv --python='/usr/local/bin/python3.7' virtualenv
# Entramos al entorno virtual
source virtualenv/bin/activate
# Desde el entorno virtual
pip install Pillow
# Salir del entorno virtual
deactivate
```

Tras esto habrá que añadir el código fuente que queramos de la función en la carpeta antes creada. Pero estas acciones ya se corresponden de forma más acertada con el Capítulo 5.

4.4 Instalación de OpenFaaS

En está sección se verán los diferentes pasos en la instalación, destacando los requisitos necesarios para la misma y por último se realizará una configuración básica. Si se quieren recordar los conceptos teóricos básicos visitar la Subsección 3.3.3

Hay que tener en cuenta que, la instalación y configuración para uso en nube privada es diferente a la que hay que realizar si se busca usar el sistema con nube pública. Por ejemplo, para plataforma local tendremos una herramienta para lanzar contenedores en privado, pero en plataforma remota tendremos una herramienta que usa *ssh* para enviar la información a la nube pública de cómo crear el contenedor [23].

4.4.1 Requisitos previos

Como he comentado previamente, para que el funcionamiento de la herramienta sea satisfactorio en local, se necesita de un sistema que le cree los contenedores *Kubernetes* usados para el procesamiento de la función. En este caso la herramienta permite el uso de varios, entre los cuáles está *kind* (recomendado por la organización) instalado y explicado brevemente en la Subsección 4.2.1. Otras herramientas permitidas son: *k3d*, *k3s*, *minikube* y *microk8s*.

Eso sí, la herramienta se podría instalar en el cluster de *Kubernetes* antes creado por Knative, pero para mejorar el rendimiento del ordenador (el plano de control de Knative consume mucho en ese contenedor de Docker) se creará otro *cluster* de *Kubernetes* gracias a *kind* como se puede apreciar en el comando ejecutado en el Código 4.16.

Código 4.16 Creación de un nuevo cluster de *Kubernetes*.

```
kind create cluster --image kindest/node:v1.23.3 --name openfaas-kcluster
```

Además, en el caso de esta herramienta, es necesario instalar un instalador de paquetes llamado *arkade* debido a que, para instalar la base de la herramienta dentro del *cluster* de *Kubernetes* es necesario un instalador de paquetes ya sea *Homebrew* o *arkade*. Se ha utilizado *arkade* porque sirve también para instalar el CLI de OpenFaaS.

La herramienta *arkade* ofrece un *script* para la instalación de este instalador. En el Código 4.17 se visualiza el paso a seguir para completar la instalación de este sistema instalador de paquetes.

Código 4.17 install_arkade.sh.

```
curl -SLsf https://get.arkade.dev/ | sudo sh
```

4.4.2 Instalación

La instalación de este sistema estará compuesta por 2 partes. Deberá realizarse en el siguiente orden para que el proceso sea satisfactorio:

1. La instalación de la línea de comandos de OpenFaaS: *faas-cli*.
2. La instalación del *OpenFaaS Chart* (*Pods* para el correcto funcionamiento de OpenFaaS).

Instalación de *faas-cli*

La instalación de la línea de comandos es esencial debido a que esta herramienta ayudará al desarrollador en cuanto a la creación de funciones, la construcción de las mismas y ver los registros generados por éstas. En el Código 4.18 se presentan los comandos necesarios para la instalación de esta herramienta.

Código 4.18 install_faas-cli.sh.

```
# Descarga del ejecutable
arkade get faas-cli

#Mover a la carpeta de las aplicaciones
sudo mv /home/juaparcla/.arkade/bin/faas-cli /usr/local/bin/
```

Instalación de *OpenFaaS Chart*

La instalación de esta parte es estrictamente necesaria debido a que es en si la base para que OpenFaaS pueda crear funciones, desarrollar su sistema de auto-escalado y que se puedan realizar peticiones a los *Pods* que contengan las funciones. En el Código 4.19 se presenta el comando necesario para la instalación de esta herramienta.

Código 4.19 install_OpenFaaS-Chart.sh.

```
#Es necesario tener encendido el motor de Docker y el contenedor con Kubernetes
     dedicado activo
arkade install openfaas
```

4.4.3 Configuración

En esta subsección se verán los apartados necesarios para poder comprobar que la herramienta fue implementada correctamente y si es necesario cambiar algún valor de algún parámetro de la misma para que la implementación básica a realizar funcione de manera correcta.

Para comprobar el estado de la herramienta se verán los siguientes pasos en el Código 4.20 que se presenta a continuación. En este caso, las comprobaciones son muy importantes, ya que la herramienta tarda un tiempo en quedar completamente estabilizada y con todos sus elementos en funcionamiento (unos dependen de otros).

Código 4.20 Comprobación de que los contenedores asociados estén en correcto funcionamiento.

```
# Ver los contextos para seleccionar
kubectl config get-contexts

# Cambio de cluster de Kubernetes
kubectl config use-context kind-openfaas-kcluster

# Visualización de estado de las unidades mínimas y deployments de Kubernetes (
    running o disponible como estados válidos)
kubectl get pods -n kube-system
kubectl get deployments -n kube-system
```

```
# Comprobación de las unidades mínimas y deployments asociados con la instalaci
    ón del motor de OpenFaaS (son todos deployments pero con un pod asociado,
    comprobar que exista este y este en funcionamiento)
kubectl get pods -n openfaas
kubectl get deployments -n openfaas

# Comprobación de las unidades mínimas y deployments asociados al local-path-
    storage
kubectl get pods -n local-path-storage
kubectl get deployments -n local-path-storage
```

Es necesario aclarar, que al igual que OpenWhisk, el CLI de OpenFaaS necesita iniciar sesión para que pueda configurar las funciones. Esto es debido a que usa HTTP y autenticación HTTP en la comunicación con su Gateway.

Para completar la configuración de la herramienta es necesario ejecutar los siguientes comandos presentes en el Código 4.21. Estos sirven para iniciar sesión con la línea de comandos (pudiendo así crear funciones y describirlas) y abrir la conexión con el cluster de *Kubernetes* ya que si no se abre, el acceso a funciones no es posible.

Código 4.21 Comandos y ficheros necesarios para configurar de forma básica OpenFaaS.

```
# Asociar el puerto 8080 de Ubuntu al 8080 interior del cluster de Kubernetes
    para hacer alusión al service que gestiona el acceso al Gateway de OpenFaaS
    . Este comando es necesario ejecutarlo cada vez que se reinicia el cluster
    de Kubernetes asociado a OpenFaaS.
kubectl port-forward -n openfaas svc/gateway 8080:8080

# Obtener contraseña de admin creada por defecto en la instalación de OpenFaaS
kubectl get secret -n openfaas basic-auth -o jsonpath="{.data.basic-auth-
    password}" | base64 --decode; echo

# Iniciar sesión con la faas-cli
faas-cli login --username admin --password <resultado_anterior>
```

4.5 Instalación de OpenFunction de QingCloud

En está sección se verán los diferentes pasos en la instalación, destacando los requisitos necesarios para la misma y para finalizar se realizará una configuración básica de la herramienta. Si se quieren recordar los conceptos teóricos básicos visitar la Subsección 3.3.4

Esta herramienta es algo más simple que las demás por tanto, los procesos de instalación y configuración son más sencillos respecto a las demás herramientas vistas previamente [24].

4.5.1 Requisitos previos

La instalación consta de 4 requisitos, casi todos ellos ya han sido realizados en anteriores secciones, pero es importante clarificarlos antes de pasar a la instalación bruta. Éstos son:

- El primero es instalar una herramienta que sea capaz de crear *clusters* de *Kubernetes*. En anteriores instalaciones se vio el uso de *kind*, por lo que para no complicar más la instalación se va a continuar usando este.
- El segundo requisito es instalar Docker Engine aunque ya tengamos Docker Desktop. Esto se debe a que OpenFunction necesita establecer conexión total con el ordenador donde se ejecutan los contenedores. Anteriormente en las otras herramientas sólo era necesario establecer conexión con un *Gateway* o similar en la que la herramienta se basaba para conectar exterior y funciones. En esta herramienta no ocurre esto y Docker Desktop no implementa conectividad directa (entrada en la tabla encaminamiento) ya que trabaja cómo máquina virtual. En el Código 4.22 se presentan los pasos para instalar Docker Engine.

- El tercer requisito se basa en la mejora del rendimiento del ordenador y que las pruebas sean menos propensas a errores, se creará el contenedor de *Docker* (anteriormente mencionado) con un *cluster* de *Kubernetes* como se vió en la Subsección 4.4.1 pero cambiando el nombre del conjunto para poder distinguirlo de los demás, el nombre será *openfunctions-kcluster* (Código 4.23). Este punto es aún más necesario que en otros casos ya que OpenFunction trabaja instalando las dependencias asociadas directamente en contenedores de *Kubernetes*, ya sean *Pods* o *Deployments*. Estás dependencias son:

  1. Dapr.
  2. Keda.
  3. Knative Serving.
  4. Shipwright.
  5. Kourier.
  6. Ingress Controler.

- El cuarto de ellos es necesario para la propia instalación y consiste en conseguir el instalador de paquetes para *Kubernetes helm*, esta instalación se realizó en la Subsección 4.3.1.

Código 4.22 install_docker_engine.sh.

```
# Instalación de dependencias (probablemente ya instaladas)
sudo apt-get update

sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

# Añadir la clave GPG oficial de Docker (ya hecho en la instalación de Docker
    Desktop, repetir por si se hubiera actualizado)
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
     /etc/apt/keyrings/docker.gpg

# Comando para poder usar el repositorio oficial (ya hecho en la instalación de
     Docker Desktop, repetir por si se hubiera actualizado)
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.
      gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /
      dev/null

# Instalación del motor
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-
    plugin

# Paramos el motor recién instalado
sudo systemctl stop docker docker.socket containerd

# Cuando se quiera iniciar el motor ejecutar
sudo systemctl start docker docker.socket containerd

# Deshabilitamos el inicio con el sistema, para así poder elegir entre Docker
    Desktop o Docker Engine
sudo systemctl disable docker docker.socket containerd
```

En esta sección se debe trabajar con el usuario *root* para mayor comodidad, ya que al usar el motor en bruto es necesario acceder a archivos que no tienen permisos asociados a los usuarios regulares.

Código 4.23 Creación de un nuevo cluster de *Kubernetes*.

```
kind create cluster --image kindest/node:v1.23.3 --name openfunction-kcluster
```

4.5.2 Instalación

La instalación de OpenFunction consiste en 2 pequeñas partes, la instalación de su herramienta de línea de comandos (esta no es estrictamente necesaria, pero ayudará en el proceso de creación de funciones), y a partir de *helm*, la instalación del motor base para la herramienta. En los Códigos 4.24 y 4.25 se pueden ver los comandos necesarios para cumplir estas pasos del proceso de instalación.

Código 4.24 install_OFunctionCLI.sh.

```
# Descarga, descompresión y movimiento del ejecutable del CLI
wget -c https://github.com/OpenFunction/cli/releases/latest/download/ofn_linux_
    amd64.tar.gz -O - | tar -xz
chmod +x ofn && sudo mv ofn /usr/local/bin/
```

Código 4.25 install_OFunction.sh.

```
# Añadir el repositorio de OpenFunction a helm
helm repo add openfunction https://openfunction.github.io/charts/
helm repo update

# Instalación del motor, para poder instalar, Kubernetes debe estar activo (El
    contenedor con Kubernetes dedicado a OpenFunctions)
kubectl create namespace openfunction
helm install openfunction openfunction/openfunction -n openfunction --version
    0.1.0

# Si se ha producido un error, desinstalar con el siguiente comando
helm uninstall openfunction -n openfunction
```

Es interesante comentar, que también se puede instalar la herramienta mediante su línea de comandos propia (*ofn*). Eso sí, esta forma de instalación está en desuso. Las instalaciones han sido probadas de ambas maneras y se completan correctamente, por ello se ha añadido el comando para realizarlo de esta forma, si se creyese conveniente, en el Código 4.26.

Código 4.26 Comando en desuso de instalación de OpenFunction.

```
ofn install --all
```

Como última anotación sobre la instalación de la herramienta, existe una versión 0.8 (lanzada en octubre de 2022) que ya se puede usar. Eso sí, una gran parte de la documentación todavía no ha sido actualizada (las páginas de *GitHub* que introducen a la creación de funciones, por ejemplo) y es importante, ya que se cambió el funcionamiento de la herramienta en tareas cómo el acceso al *cluster* desde el exterior y que se cambiaron algunas dependencias con respecto a la versión que se está usando, la 0.6.

4.5.3 Configuración

En esta subsección, en primer lugar, se revisará si la instalación ha sido realizada correctamente, revisando el *cluster* dedicado a esta herramienta. Para ello, se comprobarán si los *deployments* y los *pods* generados

están en funcionamiento. En el Código 4.27 se muestran los comandos a ejecutar para que estas tareas se completen.

Código 4.27 Comprobación de que los contenedores de OpenFunction estén en correcto funcionamiento.

```
# Ver los contextos para seleccionar
kubectl config get-contexts

# Cambio de cluster de Kubernetes
kubectl config use-context kind-openfunction-kcluster

# Visualización de estado de las unidades mínimas y deployments de Kubernetes (
    running o disponible como estados válidos)
kubectl get pods -n kube-system
kubectl get deployments -n kube-system

# Comprobación de las unidades mínimas y deployments asociados con la instalaci
    ón del motor de OpenFunctions (son todos deployments pero con un pod
    asociado, comprobar que exista este y esté en funcionamiento)
kubectl get pods -n openfunction
kubectl get deployments -n openfunction

# Comprobación de las unidades mínimas y deployments asociados a las
    dependencias de las que necesita OpenFunctions para su funcionamiento
#Dapr:
kubectl get pods -n dapr-system
kubectl get deployments -n dapr-system
#Keda:
kubectl get pods -n keda
kubectl get deployments -n keda
#Knative Serving:
kubectl get pods -n knative-serving
kubectl get deployments -n knative-serving
#Shipwright:
kubectl get pods -n shipwright-build
kubectl get deployments -n shipwright-build
#Kourier:
kubectl get pods -n kourier-system
kubectl get deployments -n kourier-system
#Ingress
kubectl get pods -n ingress-nginx
kubectl get deployments -n ingress-nginx

# NOTA: algunos pods pueden tardar en entrar en funcionamiento ya que dependen
    de otros que se están ejecutando aún (tener esta nota en cuenta también al
    reiniciar el contenedor de Docker para OpenFunction).
```

A continuación, se creará uno de los objetos necesarios para que la herramienta este completamente configurada, este se trata de un *secret* de *Kubernetes*. Este tipo de objetos de *Kubernetes* guardan credenciales (usuario y contraseña) de algún lugar web, que también hay que especificar, para que los contenedores contenidos en el *cluster* tengan la capacidad de acceder a este sitio web sin restricciones y así no sea necesario especificar en el código de una aplicación algún tipo de contraseña o *token* sensible [19].

En el Código 4.28 que aparece a continuación, se presenta el comando necesario que hay que ejecutar para conseguir un *secret* para Docker Hub.

Código 4.28 Creación de un secret para Kubernetes.

```
kubectl create secret docker-registry push-secret \
 --docker-server="https://index.docker.io/v1/" \
 --docker-username="<usuario_docker_hub>" \
 --docker-password="<password>"
```

Para poder acceder a las funciones deberemos conocer la IP externa del contenedor y además configurar Kourier y Knative Serving para que se use esta y se puedan traducir correctamente los nombres asignados a cada función. En el Código 4.29 que aparece a continuación se verá la forma de hacer esto posible.

Código 4.29 Configuración de acceso desde el exterior en OpenFunction.

```
# Obtener la ip del contenedor (la del objeto JSON "Containers" (sólo habrá uno
     al estar utilizando Engine en vez que Desktop) en mi caso 172.18.0.2/16)
docker network inspect kind

# Asignar esta IP externa a Kourier
kubectl patch svc -n kourier-system kourier \
  -p '{"spec": {"type": "LoadBalancer", "externalIPs": ["172.18.0.2"]}}'

# Asignarla también a Knative Serving
kubectl patch configmap/config-domain -n knative-serving \
  --type merge --patch '{"data":{"172.18.0.2.sslip.io":""}}'
```

5 Creación de una función en varias herramientas de *Serverless Computing* para uso privado

El principal enemigo de la creatividad es el buen gusto.

Pablo Picasso

En este capítulo se comentarán las características en común de la función de prueba que se va a usar. Además, se creará esta función de prueba con sus particularidades en cada una de las herramientas instaladas en el Capítulo 4.

Antes de comenzar, hay que comentar que este capítulo está enfocado más al desarrollador de una empresa que tenga instalada una de las herramientas presentadas y que quiera pasar su código Python a una función FaaS. Este trabajo se centrará principalmente en crear funciones que son accedidas mediante HTTP POST.

5.1 Características de la función de prueba

En esta sección se destacarán las diversas características en común que tendrá la función en las distintas herramientas que se verán próximamente. Se verá el lenguaje de programación en el que se realizarán las pruebas y en que consistirá el algoritmo a probar. También se presentará cómo se probará la función, ya que al ser la misma aplicación deberían ser capaces de ser probadas de la misma forma.

5.1.1 Lenguaje de programación

El lenguaje de programación que se ha optado por usar es *Python 3* ya que es compatible de forma nativa con todas las herramientas presentadas y permite un desarrollo simple y sencillo apoyado en gran medida por la comunidad que ostenta detrás.

Además, es compatible con la librería para procesamiento de imágenes *OpenCV* que implementa una serie de funciones que permiten, por ejemplo cambiar a escala de grises la imagen procesada o aplicar un filtrado para dar un efecto borroso y suave a una imagen.

5.1.2 Funcionamiento

El algoritmo de ejemplo que se busca implementar para comprobar el funcionamiento de las diversas herramientas es uno que sea capaz de convertir imágenes simples, en formato JPEG por ejemplo, que tengan un fondo blanco molesto que no case con el formato de documento que se está realizando, en imágenes en formato PNG (permiten canal alfa transparente) sin ningún tipo de fondo.

Se ha elegido esta funcionalidad simplemente por el hecho de que en el desarrollo del grado, se realizan una cantidad importante de trabajos que requieren de presentaciones. En estas presentaciones se usan muchas imágenes para que éstas sean más atractivas para el que atiende. Este algoritmo, que se lanzaría mediante una simple petición HTTP, puede ser interesante ya que ahorraría gran cantidad de tiempo a las personas que lo necesiten.

En el Código 5.1 que aparece a continuación, se puede apreciar la función que usará la implementación. La función recibe cómo parámetro un objeto imagen de la librería OpenCV y devolverá otro objeto imagen con el resultado aplicado para que se use en la acción principal. La implementación en cada herramienta usará esta función cuando tenga preparado el objeto imagen de OpenCV. El algoritmo en sí viene comentado paso por paso en el fichero. El código está basado en ejemplos de la documentación de la librería [27].

Código 5.1 quitaFondo.py.

```
import numpy as np
import cv2

def quitaFondo(img_src):
    # Convertir la imagen a escala de grises
    img_gray = cv2.cvtColor(img_src, cv2.COLOR_BGR2GRAY)

    # Creamos una imagen con histiéresis para eliminar ciertos valores de
        grises que será el canal alfa
    img_whiteless = cv2.threshold(img_gray, 250, 255, cv2.THRESH_BINARY)[1]
    img_whiteless = 255 - img_whiteless

    # Para perfeccionar los bordes de esta imagen aplicamos un filtro de blur (
        hace partes borrosas)
    img_whiteless = cv2.GaussianBlur(img_whiteless, (0,0), sigmaX=2, sigmaY=2,
        borderType = cv2.BORDER_DEFAULT)

    # Estiramiento lineal, los valores a 127 se convierten en 0 (255 se
        mantiene en 255)
    img_whiteless = (2*(img_whiteless.astype(np.float32))-255.0).clip(0,255).
        astype(np.uint8)

    # Montamos la imagen resultado
    result = img_src.copy()
    result = cv2.cvtColor(result, cv2.COLOR_BGR2BGRA)
    result[:, :, 3] = img_whiteless

    return result
```

Para probar el algoritmo, antes de implementarlo en una de las herramientas, se ha realizado el *script* de prueba presente en el Código 5.2 con resultados satisfactorios en imágenes con fondo blanco (con fondos de otro tipo el algoritmo no es funcional).

Código 5.2 pruebaLibreria.py.

```
import quitaFondo as qf
import cv2

# Recogemos la imagen
img = cv2.imread('beti.jpg')

# Procesamos y recogemos
img_res = qf.quitaFondo(img)

# Guardamos en un archivo
cv2.imwrite('sinFondo.png', img_res)
```

El resultado presentado por el algoritmo es el siguiente. En primer lugar, en la Figura 5.1 se muestra una captura de la imagen abierta por un lector de imágenes con su fondo blanco. A continuación, en la

Figura 5.2 se visualiza una captura de la imagen abierta por un lector de imágenes pero esta vez con el canal alfa transparente representado.

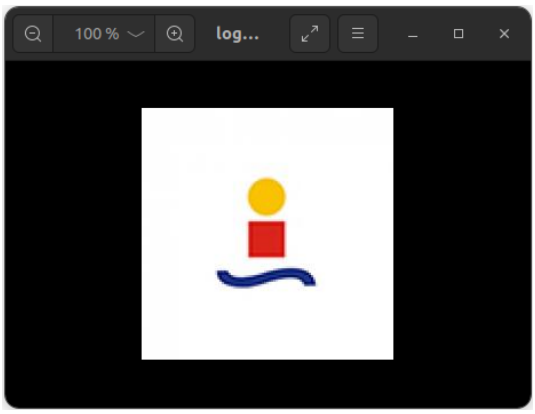

Figura 5.1 Imagen con el fondo blanco.

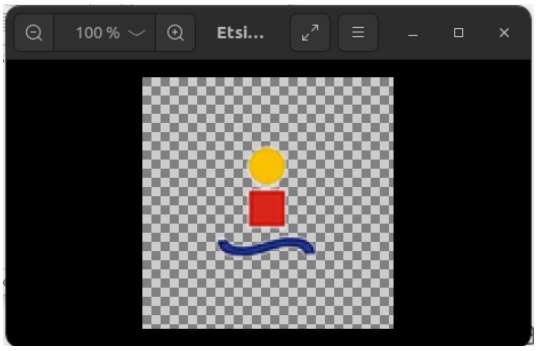

Figura 5.2 Imagen sin fondo y con canal alfa.

5.1.3 Pruebas básicas a la función

Las pruebas deberán ser independientes de la herramienta que se esté tratando, y como ya he comentado, se busca que la función implementada responda ante una petición HTTP de tipo POST con una imagen de formato JPEG en su interior, por tanto con utilizar una herramienta cómo *curl* debería ser suficiente. La petición tendrá generalmente la forma presente en el Código 5.3, si es necesario algún cambio se notificará en la sección pertinente.

Código 5.3 Peticiones mediante curl.

```
curl -F "image=@<ruta_imagen>.jpg" <url_acceso a la función> -o <ruta_imagen_
    saliente>.png
```

5.2 Creación de una función en Knative

En esta sección se verán los pasos necesarios para tener la herramienta disponible y poder implementar la aplicación a través de la parte de *serving* de Knative, la cual es la que más se adecua al propósito visto para la implementación a realizar, ya que esta parte responde a mensajes HTTP de tipo POST. También se presentarán los pasos a seguir para lanzar la función en Knative [21].

En primer lugar debemos tener la parte de *serving* instalada y tener una cuenta de Docker Hub para ser capaces de asociar nuestra imagen de contenedor personalizada de Python, a controlar por Knative, a Docker. Para personalizar una imagen se usan los archivos *Dockerfile*, en el caso que se está presentando, el contenido de este fichero se puede ver en el Código 5.4 que se presenta a continuación.

Código 5.4 Dockerfile para la implementación en Knative.

```
FROM python:3

ENV PYTHONUNBUFFERED True

COPY kn-quitafondo.py /app/

WORKDIR /app

RUN pip install Flask
RUN pip install opencv-python-headless

EXPOSE 8080

ENTRYPOINT [ "python" ]

CMD [ "kn-quitafondo.py" ]
```

En este fichero se selecciona la imagen de Python con *Debian Linux* de fondo y se ordena copiar toda la carpeta a una carpeta del futuro contenedor que se asigna como directorio de trabajo. A continuación, se instalan las diversas dependencias para Python y se asigna el puerto 8080 para escuchar peticiones. Por último, se indica que *script* hay que ejecutar.

Será necesario también un fichero de configuración para la aplicación lanzada, este es de formato YAML, y se puede ver en el Código 5.5, el cuál ha sido basado completamente en la documentación y los ejemplos de la herramienta.

Código 5.5 config-app.yaml.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: kn-quitafondo
  namespace: default
spec:
  template:
    spec:
      containers:
        - image: docker.io/erjuanpc/kn-quitafondo
```

En este fichero se definen los distintos objetos necesarios para que *serving* funcione correctamente (el objeto *Service*) usando parámetros y tipos por defecto.

Tras la generación de estos ficheros, ya se está preparado para generar la aplicación en Knative. Los comandos necesarios para generar y lanzar la función se presentan a continuación.

En primer lugar, el Código 5.6 que aparece a continuación es lo que se ejecutará en el *Pod* tras recibir el evento de HTTP POST. Este código está auto-comentado.

Código 5.6 kn-quitaFondo.py.

```
from flask import Flask, request, make_response, redirect
import numpy as np
import cv2
import os

app = Flask(__name__)

def quitaFondo(img_src):
```

```
    ...
    omitido por ser igual que en Código 5.1
    ...

@app.route('/', methods=['POST'])
def kn_quitafondo():

    if len(request.files.to_dict(flat=True))==0:
            return make_response({"msg": "No has enviado fichero con la imagen
                "})

    else:
        # Recoger imagen de la petición
        files = request.files.to_dict(flat=True)
        img_file = files['image']
        img_file.save(os.environ['HOME'] + "/" + img_file.filename)
        img = cv2.imread(os.environ['HOME'] + "/" + img_file.filename)

        # Procesar la imagen
        res_img = quitaFondo(img)

        # Codificamos la imagen en png
        res_array_img = cv2.imencode('.png',res_img)[1] # [0] es un bool que
            indica el resultado de la operación

        # Preparamos la respuesta con la imagen
        response = make_response(res_array_img.tobytes())
        response.mimetype = 'image/png'
        response.status = '200'

        return response

if __name__ == '__main__':
    app.run(debug=True,host='0.0.0.0',port=8080)
```

En principio la función *quitaFondo* iba a ir en otro fichero, pero hacer esto implicaba errores en la creación de los contenedores auxiliares ya que por algún motivo no se importaba bien dentro de la función principal.

Tras tener este código preparado, será necesario realizar los siguientes pasos para completar el proceso de creación de la función:

1. Lanzar el contenedor al repositorio online de Docker con el código creado para que Knative lo pueda usar.
2. Aplicar los valores de configuración mediante el CLI de Kubernetes.
3. Realizar las diversas comprobaciones para ver que se ha lanzado correctamente.

Lanzamiento del contenedor

Para construir el contenedor, desde el directorio de trabajo donde se sitúan el *script* y el *Dockerfile*, se deben ejecutar los siguientes comandos, desde una terminal de Ubuntu, presentes en orden en el Código 5.7 que aparece a continuación.

Código 5.7 Lanzamiento del contenedor para la app de Knative.

```
# Generar el contenedor en local
docker build -t <user_DockerHub>/kn-quitafondo . #Sustituir <> por el usuario
    de DockerHub a usar
# Situar el contenedor en el registro de Docker
docker push <user_DockerHub>/kn-quitafondo
```

Aclarar que será necesario tener encendido el motor de Docker Desktop porque si no el comando *docker* no funcionará aunque no implique acciones con contenedores locales (tipo *run*, *start* o *stop*).

Aplicar los valores de configuración

Los valores contenidos en el fichero YAML del Código 5.5 se aplicarán ejecutando la siguiente orden contenida en el Código 5.8 que se presenta a continuación con la herramienta de línea de comandos de *Kubernetes*. Para que estos comandos al menos se inicien, hay que asegurarse que está activo el contenedor que contiene el *cluster* de *Kubernetes* con Knative en su interior.

Código 5.8 Imposición de la configuración (creación de objetos).

```
# Ver los contextos para seleccionar (por si se está en otro)
kubectl config get-contexts

# Seleccionar el cluster a usar (seleccionar Knative)
kubectl config use-context kind-knative

# Aplicar la configuración
kubectl apply -f config-app.yaml

# Si se quiere borrar la app
kubectl delete -f config-app.yaml
```

Si no se crea correctamente es debido a que no se ha terminado de iniciar o se ha iniciado mal algún *pod* de Knative o *Kubernetes*. En el Código 5.9 se presentan los comandos que ayudan a visualizar el estado de estos contenedores. Antes hay que asegurarse de haber seleccionado el *cluster* indicado con el primer comando de el Código 5.8.

Código 5.9 Comandos útiles para comprobar si Knative se inició correctamente.

```
# Para ver los deployments
kubectl get deployments --all-namespaces

# Para ver las unidades mínimas
kubectl get pods --all-namespaces
```

Comprobaciones de lanzamiento

Para comprobar que la aplicación se lanzó de forma correcta, es necesario comprobar que los objetos especificados en el fichero de configuración (Código 5.5) se han lanzado correctamente y si están preparados para el funcionamiento. Para ello, en el Código 5.10 se presentan los comandos de la herramienta CLI de *Kubernetes* a lanzar. Estos sirven para comprobar si la creación de los diversos contenedores de diferentes tipos en el *cluster* de Knative ha sido satisfactoria.

Código 5.10 Comprobaciones lanzamiento de la implementación en Knative.

```
# Solo se ha creado un service por tanto para comprobar que está en
    funcionamiento
kubectl -n default get svc

# Este service tendrá un deployment asociado que se creará (estará registrado
    en la lista pero estará desactivado) cuando se realice una petición y que a
     su vez activará los pods, para comprobarlo en tiempo real ejecutar
kubectl -n default get deployments --watch

# Para ver si se activa el pod (contiene un contenedor con la imagen de docker
    que se creó en configuración) al activar el deployment
```

```
kubectl -n default get pods --watch

# Para comprobar si ha fallado la ejecución del pod anterior
kubectl describe pod > <Ruta del fichero para guardar la ejecución (resultado
    muy largo)>
```

Para obtener la url, necesaria para acceder al servicio y que se active el contenedor *deployment*, es necesario ejecutar el Código 5.11 que aparece a continuación.

Código 5.11 Recogida de URL del servicio de Knative.

```
kubectl get ksvc kn-quitafondo --output=custom-columns=NAME:.metadata.name,URL
    :.status.url

Resultado:
NAME          URL
kn-quitafondo http://kn-quitafondo.default.127.0.0.1.sslip.io
```

Por tanto para probar si se ha implementado correctamente la aplicación, será necesario ejecutar el Código 5.12, el cuál es la petición genérica antes presentada pero cambiando los valores para la prueba, ejecutada desde la carpeta donde se encuentra la imagen *beti.jpg* y se devolverá la imagen procesada (sin ningún tipo de fondo) *BetiSF.png*.

Código 5.12 Petición curl Knative.

```
curl -F "image=@beti.jpg" http://kn-quitafondo.default.127.0.0.1.sslip.io -o
    BetiSF.png

Resultado:
  % Total    % Received % Xferd Average Speed   Time    Time     Time  Current
  %          %          %       Dload  Upload  Total   Spent    Left  Speed
100 2104k 100 1443k  100  661k   517k    236k 0:00:02 0:00:02 --:--:-- 753k
```

5.3 Creación de una función en OpenWhisk

En está sección se verá como se ha realizado la creación de una función básica mediante la visualización del código final de la acción y los pasos necesarios para que sea posible crearla y alcanzable desde *curl* para su prueba.

Lo primero que se va a exponer es el código que ejecutará la acción. La principal diferencia con el presentado en la implementación en Knative es que no será necesario usar la librería *Flask* de Python para generar un servidor web, esto es debido a que esta parte de la tarea la realizará OpenWhisk [22].

En el Código 5.13 se puede ver la función que se ejecutará cuando se invoque la acción. También está presente la función que quita el fondo a la imagen pero usando la librería de procesamiento de imágenes Pillow [28].

OpenWhisk codifica en Base64 los archivos binarios enviados a las *webs actions* y necesita también devolver en Base64 los ficheros binarios que se quieran responder.

Hay que tener en cuenta que esta herramienta no parece estar demasiado preparada para ser usada con procesamiento de ficheros binarios de gran tamaño ya que presenta un límite (sin ninguna forma de cambiarlo, según la documentación) en petición y respuesta de 1MB.

Por ende, habrá que tener cuidado al enviar imágenes muy pesadas ya que la herramienta no será capaz de procesarlas, bien por el simple hecho de ser muy grandes en entrada o en salida al finalizar su procesamiento.

Código 5.13 __main__.py.

```
import sys
import io
import base64 as b64
from PIL import Image

def quitaFondo(img):
  img = img.convert("RGBA") # Cambiamos la imagen a una RGB pero con capa de
      transparencia (4x8 bits cada pixel)

  img_seq = img.getdata() # Recogemos la secuencia de píxeles de la imagen

  img_res = [] # Variable para guardar la imagen modificada
  for pixel in img_seq:
    if pixel[0] >= 245 and pixel[1] >= 245 and pixel[2] >= 245: # El blanco
        puro es 255,255,255 pero si se le aplica algo de histéresis sale un
        mejor resultado
      img_res.append((255,255,255,0)) # Pixel con la capa transparente a 0
    else:
      img_res.append(pixel)

  img.putdata(img_res) # Ponemos la nueva secuencia de pixeles
  return img


def main(dict):
  res = {}

  if '__ow_method' in dict: # Se trata de una web action
    if (dict['__ow_method']=='post'): # Método POST, puede haber imagen
      headers = dict['__ow_headers']
      content_t = headers['content-type'].split(';')[0]

      if (content_t =='multipart/form-data'): # Petición con curl -F y web=true
        raw_req = dict['__ow_body']
        raw_req_decoded = b64.decodebytes(raw_req.encode('utf-8'))
        raw_req_lines = raw_req_decoded.split(b'\n')

        # Recoger la imagen (las 5 primeras líneas contienen información del
            fichero enviado)
        raw_bytes_content = b'\n'.join(raw_req_lines[4:len(raw_req_lines)-1])

        # Procesar la imagen que viene en Base64 (sin terminar)
        image_obj = Image.open(io.BytesIO(raw_bytes_content))

        image_obj = quitaFondo(image_obj)

        img_bytes_end = io.BytesIO()
        image_obj.save(img_bytes_end, 'PNG')

        # Devolvemos la imagen codificada de nuevo en Base64
        img_result_b64 = b64.b64encode(img_bytes_end.getvalue()).decode('utf-8')

        res = {"statusCode": 200,
        "body": img_result_b64,
        "headers": {"Content-Type": 'image/png'}
        }
```

```
      else:
        res = {"statusCode": 200,
        "body": 'La imagen debe ser jpeg.\n' + headers['content-type'] + '\n'
        }

    else:
      res = {"statusCode": 200,
      "body": 'Se debe enviar una imagen mediante el método POST de HTTP.\n'
      }

  else:
    res = {"statusCode": 200,
    "body": 'No se ha creado una web action.\n'
    }

  return res
```

Para crear la acción será necesario realizar ciertos pasos, éstos comandos se presentan en el Código 5.14 que aparece a continuación. Para acciones que modifican el *runtime* en algo, el desarrollador que realiza la guía recomienda la creación mediante el CLI de OpenWhisk, en vez de usar ficheros YAML y la herramienta *wskdeploy* [26].

Código 5.14 Pasos para la creación de una acción en OpenWhisk.

```
# Comprimir el entorno virtual modificado junto a los ficheros .py necesarios
    (__main__.py)
cd envVir
zip -r envVir.zip virtualenv/ __main__.py

# Crear la acción mediante el uso de la herramienta de línea de comandos y que
    sea accesible via web
wsk -i action create ow-quitafondo --kind python:3 --main main envVir.zip --web
     true

# Para eliminar la action si se ha detectado algún error
wsk -i action delete ow-quitafondo
```

Para ejecutar el código de la función será necesario conseguir el URL local de la misma. En el Código 5.15 se visualiza el comando necesario para conseguir este objetivo.

Código 5.15 Comando para obtención de URL de acción en OpenWhisk.

```
wsk -i action get ow-quitafondo --url
```

Si la acción diese error en tiempo de ejecución, OpenWhisk ofrece un sistema de *logs* basado en que, cada activación de acción lleva consigo un *hash* que la identifica de forma inequívoca. Gracias a esto OpenWhisk guarda toda la información expulsada por la misma en su base de datos y es accesible por la API de la herramienta mediante *curl* o mediante la herramienta de línea de comandos *wsk*. En el Código 5.16 se pueden ver estos comandos que nos permiten ver los *logs* expulsados por las acciones.

Código 5.16 Comandos para comprobar errores de acciones en tiempo de ejecución.

```
# Conseguir la lista de activaciones (se actualiza cada vez que se llama al
    comando. Si una activación ya terminada no aparece, volver a ejecutar)
wsk -i activation list

# Ver los logs de una cierta activación
```

```
wsk -i activation logs <id_activación>

# Ver la información de la activación más completa (incluye logs)
wsk -i activation get <id_activación>
```

Finalmente, para terminar la subsección, se presenta en el Código 5.17 un ejemplo de ejecución de la acción mediante *curl*. El parámetro -k es estrictamente necesario (ocurre cómo con el parámetro -i de *wsk*), ya que la API de OpenWhisk está basada en HTTPS con un certificado auto-firmado y este parámetro de *curl* sirve para ignorar las advertencias de seguridad que se presentan al hacer peticiones a webs de este tipo.

Código 5.17 Petición curl OpenWhisk.

```
curl -k -F "im=@cope-logo.jpg" https://localhost:31001/api/v1/web/guest/default
    /ow-quitafondo -o prueb.png

Resultado:
  % Total    % Received % Xferd Average Speed    Time    Time     Time   Current
  %          %          %       Dload  Upload   Total   Spent    Left    Speed
100  174k  100  148k   100 26172 16197   2783  0:00:09  0:00:09 --:--:-- 36299
```

5.4 Creación de una función en OpenFaaS

En esta sección se presentarán los pasos necesarios para crear la función y que ésta funcione correctamente. Se probará usando un cliente HTTP cualquiera que permita enviar archivos al destino (se ejemplificará con *curl* como ya se expuso) [23].

Para ayudar en la creación de cualquier tipo de función, la herramienta de línea de comandos de OpenFaaS trae consigo una funcionalidad que crea los ficheros base necesarios para poder lanzar la futurible función. En el Código 5.18 que aparece a continuación, se ve el comando necesario que hay que ejecutar para obtener este esquema predefinido.

Código 5.18 Comando para crear el esquema de ficheros.

```
# La template de lenguaje a usar no viene por defecto, hay que descargarla en
    local por medio de la "tienda" de templates (tambien existe la template "
    http" pero esta no nos da facilidad en cuanto a la gestión de contenido
    binario)
faas-cli template store pull python3-flask-debian

# Creación del esquema de ficheros
faas-cli new quitafondo --lang=python3-flask-debian
```

Tras esto habrá que adecuar el esquema generado cambiando valores del fichero YAML para que la parte de la subida de la imagen generada se pueda hacer (sin la variable de entorno RAW_BODY no se lee correctamente el contenido binario ya que sin ella OpenFaaS no está preparado para recibir este tipo de contenido). Además, por supuesto de crear la función en el fichero handler.py. En los siguientes Códigos (5.19, 5.20 y 5.21) se puede observar el contenido de los ficheros relatados a la función.

Código 5.19 quitafondo.yml.

```
version: 1.0
provider:
  name: openfaas
  gateway: http://127.0.0.1:8080
functions:
  quitafondo:
```

```
    lang: python3-flask-debian
    handler: ./quitafondo
    image: erjuanpc/quitafondo:latest
    environment:
      RAW_BODY: True
```

Del Código 5.19 es preciso comentar que para adaptar al usuario de Docker es necesario sustituir *erjuanpc* por la cuenta que se esté usando. Además, comentar que el lenguaje o plantilla usado es necesario que tenga la etiqueta *debian*, sin esta no se podría instalar correctamente la librería de OpenCV.

La etiqueta *flask* sirve para controlar de forma más precisa la información que se envía en la respuesta. En un principio se iba a usar con la etiqueta *http* (permite ver valores asociados a la petición), pero ésta por algún motivo no es compatible con la variable de entorno RAW_BODY.

Código 5.20 handler.py.

```
import cv2
import numpy as np

def quitaFondo(img_src):
    ...
    omitido por ser igual que en Código 5.1
    ...


def handle(req):

    raw_body_lines = req.split(b'\n')
  # Recoger la imagen (las 5 primeras líneas contienen información del fichero
      enviado. La última no contiene información)
    raw_bytes_content = b'\n'.join(raw_body_lines[4:len(raw_body_lines)-2])

    # Pasar a un objeto de imagen de OpenCV
    raw_bytes_numpy = np.frombuffer(raw_bytes_content, dtype=np.uint8)
    img = cv2.imdecode(raw_bytes_numpy, cv2.IMREAD_COLOR)

    # Llamada al procesamiento
    res_img = quitaFondo(img)

    # Codificación de la respuesta en png
    res_array_img = cv2.imencode('.png',res_img)[1] # [0] es un bool que indica
         el resultado de la operación

    return res_array_img.tobytes(), 200, {"Content-Type":"image/png"}
```

Código 5.21 requirements.txt.

```
opencv-python-headless
```

Para realizar el lanzamiento de la función a partir del fichero YAML de configuración es necesario ejecutar los diversos comandos de la CLI de OpenFaaS presentes en el Código 5.22 que se puede visualizar a continuación. Estas ejecuciones se realizan desde la carpeta donde se encuentra el fichero YAML.

Código 5.22 Comandos para lanzar en un nuevo la función a partir de su fichero YAML.

```
# Creación de la imagen para Docker
faas-cli build -f ./quitafondo.yml
```

```
# Subir la imagen de Docker generada al registro público
faas-cli push -f ./quitafondo.yml

# Desplegado (devolverá la url)
faas-cli deploy -f ./quitafondo.yml

# Los procesos de construcción, push y lanzamiento se pueden sustituir por el ú
    nico comando siguiente, pero es recomendable hacerlos de 1 en 1 para
    enfocar mejor un fallo si este se produce
faas-cli up -f ./quitafondo.yml

# Si ha habido algún error humano y la función que se ha lanzado no es la que
    queremos tener, se puede eliminar con el siguiente comando
faas-cli remove quitafondo
```

Tras realizar el despliegue, es conveniente saber algunos comandos de la CLI de OpenFaaS que pueden ayudar a saber si la función se ha creado correctamente. En el Código 5.23 presente debajo de este texto se pueden ver estos comandos.

Código 5.23 Comandos para comprobar el despliegue de una función en OpenFaaS.

```
# Lista los nombres de las funciones desplegadas
faas-cli list

# Da información relevante sobre determinada función desplegada (URL síncrono y
     asíncrono, replicas totales configuradas, nombre de la imagen usada...)
faas-cli describe <nombre_funcion>

# Imprime los logs que produzca una función en tiempo real
faas-cli logs <nombre_funcion>

# Para comprobar si la función está disponible (AVAILABLE = 1) (La librería
    OpenCV es pesada por la que función suele tardar cierto tiempo en estar
    disponible)
kubectl get deploy -n openfaas-fn -w

# Si la creación del deploy o del pod dan error
kubectl get pods -n openfaas-fn
kubectl describe deploy(pods)/<nombre_completo> -n openfaas-fn
```

Por último, y tras comprobar que el despliegue de la función ha sido satisfactorio, para realizar el proceso de prueba se ejecutará *curl* con el formato establecido en la Subsección 5.1.3. En el Código 5.24 que aparece a continuación se puede ver el comando ejecutado con el formato ya aplicado. Según la documentación, no hay límite de datos a enviar en la petición, así que se puede enviar cualquier tipo de imagen.

Código 5.24 Petición curl OpenFaaS.

```
curl -F "image=@beti.jpg" http://127.0.0.1:8080/function/quitafondo -o BetiSF.
    png

Resultado:
  % Total    % Received % Xferd Average Speed   Time    Time     Time  Current
  %          %          %       Dload  Upload  Total   Spent    Left  Speed
100 2104k 100 1443k  100  661k 2494k   1142k 0:00:02 0:00:02 --:--:-- 3640k
```

5.5 Creación de una función en OpenFunction

En esta sección se presentan los diferentes comandos, códigos y ficheros necesarios para la realización de una implementación básica, incluyendo hacer alcanzable la función desde *curl*. Una diferencia palpable respecto a anteriores herramientas es que OpenFunction trabaja usando repositorios de *Git* para el guardado del código fuente de las respectivas funciones (según la documentación la fuente de código fuente debe ser un repositorio Git y no se indica otra forma de obtener el código fuente) [24].

Cómo en otras plataformas (por ejemplo Knative), en OpenFunction la creación de funciones se basa en ficheros YAML estilo *Kubernetes*, eso sí, OpenFunction tiene su propio CRD llamado *Function* cuya documentación está presente en su página. Esto permitirá un mejor seguimiento de la creación de las funciones aunque su documentación sea escasa. En el Código 5.25 se visualiza el fichero YAML necesario para la creación de la función que se implementará en esta subsección.

Código 5.25 quitafondo.yaml.

```
apiVersion: core.openfunction.io/v1beta1
kind: Function
metadata:
  name: quitafondo
spec:
  version: "v1.0.0"
  image: "erjuanpc/quitafondo-func:v1"
  imageCredentials:
    name: push-secret
  port: 8080 # default to 8080
  build:
    builder: "openfunction/gcp-builder:v1"
    env:
      GOOGLE_FUNCTION_TARGET: "main"
      GOOGLE_FUNCTION_SIGNATURE_TYPE: "http"
      GOOGLE_FUNCTION_SOURCE: "main.py"
    srcRepo:
      url: "https://github.com/JohnStopped/ImplOpenFunctionTFG.git"
      revision: "prueba-kn-code"
  serving:
    runtime: knative # default to knative
    template:
      containers:
        - name: function # DO NOT change this
          imagePullPolicy: Always
  route:
    gatewayRef:
      name: openfunction
      namespace: openfunction
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /quitafondo
    hostnames:
    - "sample.ofn.io"
```

En cuanto al código fuente, como se comentó previamente se debe hacer uso de repositorios de GitHub, en este caso se hará mediante un repositorio público (si fuese privado sería necesario crear un token de GitHub con únicamente permisos de bajada del repositorio y además añadir un *secret* de *Kubernetes* (similar al añadido en la Subsección 4.5.3) que contenga a éste) cuya URL está presente en el Código 5.25 anteriormente presentado.

En el Código 5.26 que aparece después de este párrafo, se ve el contenido del fichero *main.py* en el que se basará la función. Sobre este código, comentar que es prácticamente igual que el código fuente utilizado con la herramienta Knative ya que OpenFunction basa el procesado de las peticiones y respuestas mediante la librería *Flask* de Python, al igual que se hizo con Knative. Este dato se ha descubierto mediante prueba y error, no aparece nada sobre esto en la documentación de la herramienta.

Código 5.26 main.py.

```
from flask import Flask, request, redirect
import numpy as np
import cv2
import os

def quitaFondo(img_src):
    ...
    omitido por ser igual que en Código 5.1
    ...


def main(request):
    if len(request.files.to_dict(flat=True))==0:
           return "No has enviado fichero con la imagen\n"

    else:
        # Recoger imagen de la petición
        files = request.files.to_dict(flat=True)
        img_file = files['image']
        img_file.save(os.environ['HOME'] + "/" + img_file.filename)
        img = cv2.imread(os.environ['HOME'] + "/" + img_file.filename)

        # Procesar la imagen
        res_img = quitaFondo(img)

        # Codificamos la imagen en png
        res_array_img = cv2.imencode('.png',res_img)[1] # [0] es un bool que
            indica el resultado de la operación

        # Preparamos la respuesta con la imagen
        response = res_array_img.tobytes()
        return response
```

Como en anteriores herramientas, también es necesario un fichero que contenga las librerías de Python necesarias para que la función funcione de forma correcta. Este es el fichero *requirements.txt* cuyo contenido está presente en el Código 5.27 que aparece a continuación. Las dos primeras librerías vienen por defecto en los ejemplos de la herramienta y son estrictamente necesarias.

Código 5.27 requirements.txt.

```
functions-framework==1.4.3
markupsafe==2.0.1
opencv-python-headless
```

Tras haber presentado todos los ficheros que componen la creación de una función en esta herramienta, se van a ver los comandos necesarios para hacer posible que ésta esté preparada en el *cluster* para recibir peticiones desde el exterior y para comprobar que la función está en funcionamiento. En el Código 5.28 que se visualiza tras este texto, se ven los comandos que se han comentado anteriormente.

Código 5.28 Comandos para crear, eliminar y comprobar la creación de funciones en OpenFunction.

```
# Creación de la función (mediante CLI o mendiante kubectl)
ofn create -f quitafondo.yaml
kubectl apply -f quitafondo.yaml

# Elimina una función
ofn delete quitafondo (nombre incluido en el objeto YAML metadata)
kubectl delete -f quitafondo.yaml

# Comprobación de cómo se desarrolla la creación de la función
kubectl get function -w
kubectl get pods --all-namespaces -w
```

Antes de proceder a la petición mediante *curl*, se debe recoger la URL que se asocia con la función. En esta herramienta la URL resultante no es tan clara cómo en las dos anteriores herramientas, y es algo más complicada de conseguir (no hay que ejecutar un único comando). En el Código 5.29 presente a continuación, se ven los comandos necesarios para obtener la URL que permite el acceso HTTP a la función.

Código 5.29 Comandos para obtener la URL en OpenFunction.

```
# Obtener el nombre del service de Kubernetes asociado a la función
kubectl get function

# Obtener la URL
kubectl get ksvc | grep <nombre_service>
```

Por último, se procede a realizar la prueba con *curl* para comprobar que la función realiza lo pedido y es accesible desde el exterior. En el Código 5.30 se visualiza el comando y su resultado.

Código 5.30 Petición curl OpenFunction.

```
curl -F "image=@beti.jpg" <url_obtenida_anteriormente> -o prueba.png

Resultado:
  % Total    % Received % Xferd Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2104k 100 1443k  100  661k   182k   85470 0:00:07 0:00:07 --:--:-- 416k
```

6 Configuración del auto-escalado y pruebas de rendimiento sobre las herramientas de *Serverless Computing* para uso privado

No tienes nada que hacer con la altura, es tu fuerza lo que cuenta.

Lynn Hill

En este Capítulo se comentarán las acciones necesarias para poner a punto un auto-escalado desde cero y máximo de iguales condiciones en las herramientas presentadas en el anterior capítulo (Capítulo 4).

Además, se realizará un análisis del rendimiento de las herramientas, en esta configuración antes comentada, apoyándose en algunas gráficas comparativas (tiempo total de ejecución con muchas peticiones, tiempo de ejecución desde cero, tiempo de ejecución con función establecida) y tablas resumen.

También, se presentarán los parámetros generales que tendrán en común todas las herramientas a analizar y que metodología seguirán las pruebas a realizar y con que *scripts* se harán.

6.1 Generalidades en la configuración y metodología de realización de las pruebas

En esta sección se buscará comentar que significan y con que valor se configurarán los parámetros de auto-escalado más interesantes para que los variados análisis salgan adelante.

También se tratará de explicar que modo de proceder se seguirá a la hora de realizar las pruebas de rendimiento a cada herramienta.

6.1.1 Metodología de realización de las pruebas

Para realizar cada una de las pruebas se creará un *script* por cada una de ellas, al cuál únicamente habrá que pasarle la *URL* que corresponda a la función de cada herramienta. En el caso de la prueba de carga, habrá que pasar por línea de comandos las iteraciones y los procesos máximos simultáneos también. Para que se complete cada prueba se deberá lanzar diez veces el *script* y así poder sacar una media de los resultados.

Para simplificar el trabajo al lector (en el caso que quiera ejecutar las pruebas), se realizará un *script* general de prueba que extraerá todos los resultados ya con la media realizada.

En las siguientes pequeñas secciones se detallan los *scripts* generales por prueba y el *script* total. Es necesario recalcar que estos *scripts* están preparados para ser ejecutados por BASH.

Prueba de carga o de tiempo total de ejecución con muchas peticiones

En esta prueba se tratará de medir cómo de rápido es capaz cada herramienta de reaccionar, usando el auto-escalado disponible, ante ráfagas de peticiones que se produzcan durante un buen periodo de tiempo. La herramienta que menos tiempo tarde en realizar la prueba puede considerarse la mejor en cuanto a reacción a un estrés de este tipo. Esta prueba es más determinante para aplicaciones que puedan sufrir este tipo de tráfico.

En el Código 6.1 que aparece a continuación está presente el *script* que realiza esta prueba.

Este código se basa en lanzar ráfagas de peticiones *curl* (número asignable con el tercer parámetro de línea) las cuales serán controladas en un inicio para ayudar al lanzamiento de los primeros contenedores. Si no se hace esto, al estar lanzándose desde cero, se perderán algunas peticiones iniciales. Esto es debido a que se asignarían muchas peticiones a la réplica inicial hasta que la herramienta se percata del alto ritmo de peticiones. Esta réplica no sería capaz de procesarlas ignorando muchas de ellas. Mientras se asignan estas peticiones a otras réplicas y éstas se crean, *curl* es posible que haya agotado el *timeout* configurado.

Al final, el *script* devuelve el tiempo de ejecución de la prueba.

Código 6.1 prueba_carga.sh.

```
URL=$1
iteraciones=$2
procesos=$3
procesos_act=1

echo "Iteraciones: $iteraciones. Procesos simultaneos máximos: $procesos.
    URLdestino: $URL"

SECONDS=0 # Variable especial de bash que sirve para contabilizar segundos
declare -a pids # Array para guardar pids intermedios y así asegurar su
    finalización
# Ejecutar la prueba
for i in $(seq 1 $iteraciones)
do
  #echo $i
  for j in $(seq 1 $procesos_act)
  do
    #echo $j
    {
    curl -k -F "image=@esti-logo.jpg" $URL -o images/$i-$j.png & #Por si se
        quiere comprobar el resultado de las imágenes. Utilizar la otra
        sentencia que no guarda imágenes si se aprecia la vida del disco duro (
        se crean algo más de 600 archivos de imagen si se usan los valores que
        se indican en el script de prueba general)
    #curl -k -F "image=@esti-logo.jpg" $URL &
    pids[${j}]=$!
    } &> /dev/null # Evita la impresión de la salida de curl
    #echo "Proceso ${pids[${j}]}."
    sleep 0.05
  done

  for pid in ${pids[*]}; do
    #echo "Esperando proceso $pid."
      wait $pid
    #echo "Terminado."
  done

  # Para empezar con baja carga y a mitad de procesado estar iterando 1 más y 1
       menos. Esto se hace para variar la carga de peticiones un poco y de vez
      en cuando. Al usar módulo 3 y módulo 6 se empezará lentamente y
      progresivamente y finalmente se llegará al número de procesos máximo
      indicado.
  if [ $(($i % 3)) -eq 0 ]; then
    if [ $procesos_act -lt $procesos ]; then
      procesos_act=$(($procesos_act + 1))
    fi
```

```
  fi
  if [ $(($i % 6)) -eq 0 ]; then
    procesos_act=$(($procesos_act - 1))
  fi

  #echo $procesos_act
  sleep 0.1
done
wait $! #Espera a todos los últimos lanzamientos

duracion=$SECONDS

#echo "La duración de la prueba han sido $duracion segundos."
exit $duracion # Devolver el valor al script general
```

Prueba de velocidad desde cero

En esta prueba se tratará de medir cómo de veloz es cada herramienta en cuanto a lanzar la primera réplica. Antes de lanzar la prueba, habrá que asegurar que no hay ninguna réplica en funcionamiento. Esta prueba será más significativa si se implementan aplicaciones con un uso bastante esporádico, tanto cómo que su uso debe ser tan bajo que compense tener el escalado desde 0.

En el Código 6.2 que aparece tras este párrafo se presenta el *script* que realiza la prueba de velocidad desde cero. Éste es mucho más simple que el anterior ya que, únicamente consistirá en una petición *curl* clásica. Eso sí, se incluirá el medidor de tiempo y la directiva *exit* para que este valor pueda ser devuelto al *script* completo.

Código 6.2 prueba_vel0.sh.

```
URL=$1
SECONDS=0 # Variable especial de bash que sirve para contabilizar segundos

{
curl -k -F "image=@esti-logo.jpg" $URL -o CopeSF.png
} &> /dev/null # Evita la impresión de la salida de curl

duracion=$SECONDS
echo "La duración de la prueba han sido $duracion segundos."
exit $duracion
```

Prueba de velocidad con función establecida

En esta prueba se buscará cuantificar qué herramienta tiene una potencia de procesado bruta más alta. En este caso no importa la situación previa de las réplicas, en cuanto si hay alguna en ejecución o no, de hecho, en el *script* se busca, de forma intencionada, que haya alguna réplica activa en el *cluster*. Esta prueba será más significativa para aplicaciones cuya tasa de petición sea de una por minuto. Con esta tasa el escalado desde cero no termina de ser rentable. También será interesante por ver cuál herramienta gestiona mejor los recursos.

En el Código 6.3 se visualiza el *script* que implementa esta prueba. Éste es prácticamente igual al de la prueba anterior, sólo que incluye una petición más a la que no se le cuenta el tiempo.

Código 6.3 prueba_vel_bruta.sh.

```
URL=$1

# Primera ejecución para asegurar que se ejecuta en una réplica creada
{
curl -k -F "image=@etsi-logo.jpg" $URL
```

```
} &> /dev/null # Evita la impresión de la salida de curl

start=$(date +%s%N)

{
curl -k -F "image=@etsi-logo.jpg" $URL -o CopeSF.png
} &> /dev/null # Evita la impresión de la salida de curl

end=$(date +%s%N)

duracion=$(($end-$start))/1000000))
echo "La duración de la prueba han sido $duracion milisegundos."
exit $duracion
```

Script completo de prueba

Este *script* sirve para simplificar el proceso de prueba en cada herramienta. Una de sus peculiaridades es que se realizarán varias pruebas (10 en este caso) para obtener un resultado más preciso e intentar evitar interpretaciones erróneas de los experimentos realizados.

En el Código 6.4 que se visualiza a continuación, aparece este *script*.

Código 6.4 prueba_total.sh.

```
URL=$1
pruebasXherr=$2

echo "Iteraciones por prueba para sacar media: $pruebasXherr. URLdestino: $URL"


# Prueba de carga
mediaCarga=0

for i in $(seq 1 $pruebasXherr)
do
  ./prueba_carga.sh $URL 100 7
  valor_devuelto=$?
  mediaCarga=$(($mediaCarga + $valor_devuelto))
  #sleep 2m # Para que los contenedores vuelvan al estado inicial
  sleep 11m # Para que los contenedores vuelvan al estado inicial en OpenWhisk
done

mediaCarga=$(echo 4 k $mediaCarga $pruebasXherr / p | dc)


# Prueba de velocidad desde 0
mediaVel0=0

for i in $(seq 1 $pruebasXherr)
do
  ./prueba_vel0.sh $URL
  valor_devuelto=$?
  mediaVel0=$(($mediaVel0 + $valor_devuelto))
  #sleep 2m # Para que los contenedores vuelvan al estado inicial
  sleep 11m # Para que los contenedores vuelvan al estado inicial en OpenWhisk
done

mediaVel0=$(echo 4 k $mediaVel0 $pruebasXherr / p | dc)
```

```
# Prueba de velocidad bruta
mediaVelBru=0

for i in $(seq 1 $pruebasXherr)
do
  ./prueba_vel_bruta.sh $URL
  valor_devuelto=$?
  mediaVelBru=$(($mediaVelBru + $valor_devuelto))
  sleep 1 # Para que los contenedores pierdan algo de carga
done

mediaVelBru=$(echo 4 k $mediaVelBru $pruebasXherr / p | dc)


# Resultados
echo "Prueba carga: $mediaCarga s"
echo "Prueba velocidad desde 0: $mediaVel0 s"
echo "Prueba velocidad bruta: $mediaVelBru ms"
```

6.1.2 Generalidades en la configuración

En todas las herramientas será necesario modificar diversos valores de auto-escalado, estos son: los valores para el escalado desde cero (activado o no, tiempo de duración de la réplica inicial...), los valores de concurrencia (cuantas réplicas a la vez pueden estar en estado de ejecución), los límites generales de escalado desde mínimos hasta máximos (réplicas en cualquier estado, es decir, en desmantelamiento, en creación o en ejecución, mínimas o máximas) y por último, el número máximo de peticiones por segundo por réplica.

Los valores usados finalmente para estos parámetros de auto-escalado serán [21][22][23][24]:

- En escalado desde cero:
  - Escalado desde cero activado. Para la prueba de velocidad desde función establecida se fuerza a generar una réplica, por ende no es necesario cambiar este valor en la configuración.
  - Tiempo de duración máximo de la réplica inicial: 15 segundos.
- En concurrencia:
  - Límite duro (número máximo de peticiones procesables por cada réplica): 0 valor por defecto que significa que no hay límite.
    En el caso de las pruebas a realizar este valor no es demasiado determinante. De hecho para la prueba de carga es mejor usar este valor ya que no se generarán nuevas réplicas por el simple hecho de repetir mucho la prueba. Así se verán mejor las capacidades de la herramienta para auto-escalar.
  - Límite suave (valor máximo de réplicas concurrentes que se puede pasar únicamente si hay una ráfaga de peticiones puntual en la vida de la función): No se usa, sirve para configurar mediante la métrica de concurrencia.
- En límites generales:
  - Límite inferior: 0 porque se usa escalado desde cero.
  - Límite superior: 150. Se usa este alto número para así asegurarnos que en la prueba de carga no se llega al límite de réplicas permitidas.
- En número máximo de peticiones por segundo que se pueden recibir por réplica: 2. Se usa este objetivo de escalado tan bajo debido al tiempo de procesado normal de la función, situado normalmente en torno a 7 segundos (si es con una nueva réplica).
  También se usa este valor porque así en la prueba de carga se mandarán a usar más réplicas y el objetivo de esta prueba es éste.

Será necesario también declarar una métrica por la que regir el auto-escalado, en nuestro caso el uso de peticiones por segundo de cada réplica. También, se podría hacer por CPU, por uso de memoria o incluso de alguna forma personalizada.

6.2 Configuración específica necesaria por herramienta

En esta sección se detallará cómo se aplican los valores de auto-escalado antes comentados a cada una de las herramientas a analizar.

6.2.1 Knative

Para aplicar los valores sobre el escalado desde cero, Knative sólo permite el uso de configuración global y se hará mediante el despliegue de un objeto ConfigMap de *Kubernetes*. El fichero YAML que configura este objeto aparece en el Código 6.5 y viene comentado con los valores sobre esta parte de la configuración [21].

Código 6.5 config-map.yaml.

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: config-autoscaler
 namespace: knative-serving
data:
 enable-scale-to-zero: "true" # Activación del escalado desde 0
 scale-to-zero-grace-period: "15s" # Tiempo máximo de duración de la réplica 0
```

Si se quieren conseguir los otros valores expuestos en la anterior subsección (límite duro, extremos inferior y superior, etc.) en Knative, se configuran directamente en el *service* de *Kubernetes* que sirve para lanzar la función. El fichero YAML que configura este *service* aparece en el Código 6.6 y únicamente cambia respecto a la implementación básica en que se añaden los valores de configuración del auto-escalado.

Código 6.6 config-app.yaml.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: kn-quitafondo
  namespace: default
spec:
  template:
    spec:
      containerConcurrency: 100
      containers:
        - image: docker.io/erjuanpc/kn-quitafondo
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "0" # Réplicas mínimas que presenta
            la función
        autoscaling.knative.dev/max-scale: "150" # Réplicas máximas en
            cualquier estado
        autoscaling.knative.dev/metric: "rps" # Selección de la métrica por
            peticiones por segundo (requests per second)
        autoscaling.knative.dev/target: "2" # Objetivo para creación de nueva r
            éplica, 2 rps a una réplica. Usamos este número bajo debido al
            tiempo de procesamiento de una función (unos 7 segundos).
```

6.2.2 OpenWhisk

OpenWhisk no permite la modificación de los valores de auto-escalado para cada función, por lo que algunos valores estarán limitados por esta herramienta, por ejemplo la concurrencia de réplicas en procesamiento a la vez (1000 réplicas) [22].

Otro valor limitado por esta herramienta es el de las activaciones/peticiones por minuto (5000). En peticiones por segundo serían unas 84.

Esta herramienta mantiene vivas las réplicas durante 10 minutos si no reciben peticiones (para cambiar esto habría que cambiar un fichero de la instalación de OpenWhisk), por tanto, en el *script* general habrá que esperar más tiempo entre pruebas.

6.2.3 OpenFaaS

Para aplicar los valores de configuración antes expuestos en OpenFaaS, simplemente habrá que añadirlos en el fichero YAML de configuración de la función. En cuanto a valores diferentes de configuración sólo aparece uno nuevo, el porcentaje de aplicación del *target* (tanto por ciento del objetivo especificado para que se creen nuevas réplicas). Para que la configuración sea igual que en Knative, hay que poner este valor con el valor expuesto (100 %) [23].

En el Código 6.7 se presenta este fichero YAML modificado para que se use el auto-escalado en esta herramienta. Eso sí, el escalado desde cero no está presente en la versión por defecto de la herramienta (habría que pagar la versión Pro), por tanto, aunque esté presente en la configuración no funcionará. Por suerte, el auto-escalado de la versión gratuita es por *RPS*.

Código 6.7 quitafondo.yml.

```
version: 1.0
provider:
  name: openfaas
  gateway: http://127.0.0.1:8080
functions:
  quitafondo:
    lang: python3-flask-debian
    handler: ./quitafondo
    image: erjuanpc/quitafondo:latest
    label:
      com.openfaas.scale.zero: True
      com.openfaas.scale.max: 150
      com.openfaas.scale.min: 0
      com.openfaas.scale.zero-duration: 1m # Que las réplicas no duren más de 1
          minuto
      com.openfaas.scale.type: rps
      com.openfaas.scale.target: 2
      com.openfaas.scale.target-proportion: 1.0
    environment:
      RAW_BODY: True
```

En el caso de esta herramienta, la prueba de velocidad desde cero no tiene sentido, por ende no aparecerá en los resultados. Además, la prueba de carga será más benévola, ya que no se perderá tiempo en generar la primera réplica, aún así, creo que es conveniente realizarla.

6.2.4 OpenFunction

En esta subsección hay que tener en cuenta que OpenFunction usa Knative Serving cómo base para ejecutar las funciones síncronas. Por tanto, en primer lugar habría que aplicar la configuración anteriormente vista en el Código 6.5. Esto es así porque en Knative es necesario configurar de forma global el uso o no del escalado desde cero.

Finalmente no será necesario ya que viene configurado por OpenFunction por defecto de la misma manera que en ese fichero YAML [24].

En segundo lugar, habrá que aplicar la configuración específica de la función. Esto se conseguirá con el fichero YAML de configuración de la misma, y al usar Knative como base, las variables a configurar son las mismas que con esta herramienta.

En el Código 6.8 se visualiza este fichero YAML previamente comentado.

Código 6.8 quitafondo.yaml.

```
apiVersion: core.openfunction.io/v1beta1
kind: Function
metadata:
  name: quitafondo
spec:
  version: "v1.0.0"
  image: "erjuanpc/quitafondo-func:v1"
  imageCredentials:
    name: push-secret
  port: 8080 # default to 8080
  build:
    builder: "openfunction/gcp-builder:v1"
    env:
      GOOGLE_FUNCTION_TARGET: "main"
      GOOGLE_FUNCTION_SIGNATURE_TYPE: "http"
      GOOGLE_FUNCTION_SOURCE: "main.py"
    srcRepo:
      url: "https://github.com/JohnStopped/ImplOpenFunctionTFG.git"
      revision: "prueba-kn-code"
  serving:
    scaleOptions:
      autoscaling.knative.dev/min-scale: 0
      autoscaling.knative.dev/max-scale: "150"
      autoscaling.knative.dev/metric: "rps"
      autoscaling.knative.dev/target: 2
    runtime: knative # default to knative
    template:
      containers:
        - name: function # DO NOT change this
          imagePullPolicy: Always
  route:
    gatewayRef:
      name: openfunction
      namespace: openfunction
    rules:
      - matches:
          - path:
              type: PathPrefix
              value: /quitafondo
    hostnames:
    - "sample.ofn.io"
```

6.3 Resultados obtenidos por las pruebas

En esta sección se expondrán, en forma de gráfico de barras (es un dato por herramienta), las diversas conclusiones extraídas de los experimentos expuestos en la Subsección 6.1.1. Además también se aclararán algunas particularidades a tener en cuenta de estos resultados.

También se tratará de comparar objetivamente, mediante el uso de una tabla (para que se vean más rápidamente los resultados), las herramientas mediante una media de los resultados de las pruebas y usando porcentajes, siendo la que mayor porcentaje obtenga la mejor de forma empírica.

6.3.1 Prueba de carga

La prueba de carga implicará que herramienta es mejor y más flexible en cuanto haya una gran ratio mantenida de peticiones por segundo. La mejor herramienta en esta prueba será la más capacitada en cuanto a velocidad del auto-escalado de forma general.

Comentar que esta prueba tiene una dependencia mínima de lo ejecutado en las réplicas, esto es debido a que la velocidad bruta de procesamiento de la función es prácticamente nula con respecto a la velocidad de creación de nuevas réplicas. Este dato se corroborará con la diferencia de las dos pruebas siguientes.

A continuación, en la Figura 6.1, se presentan los resultados de este experimento. El eje Y está presentado en segundos y tal y como dice la figura, menos tiempo implica un resultado mejor. Destacar que como se comentó en la Subsección 6.2.3, la prueba con OpenFaaS es más benevolente ya que, éste en la versión estándar no permite escalar desde cero. Los resultados presentados por la prueba son interesantes sobretodo

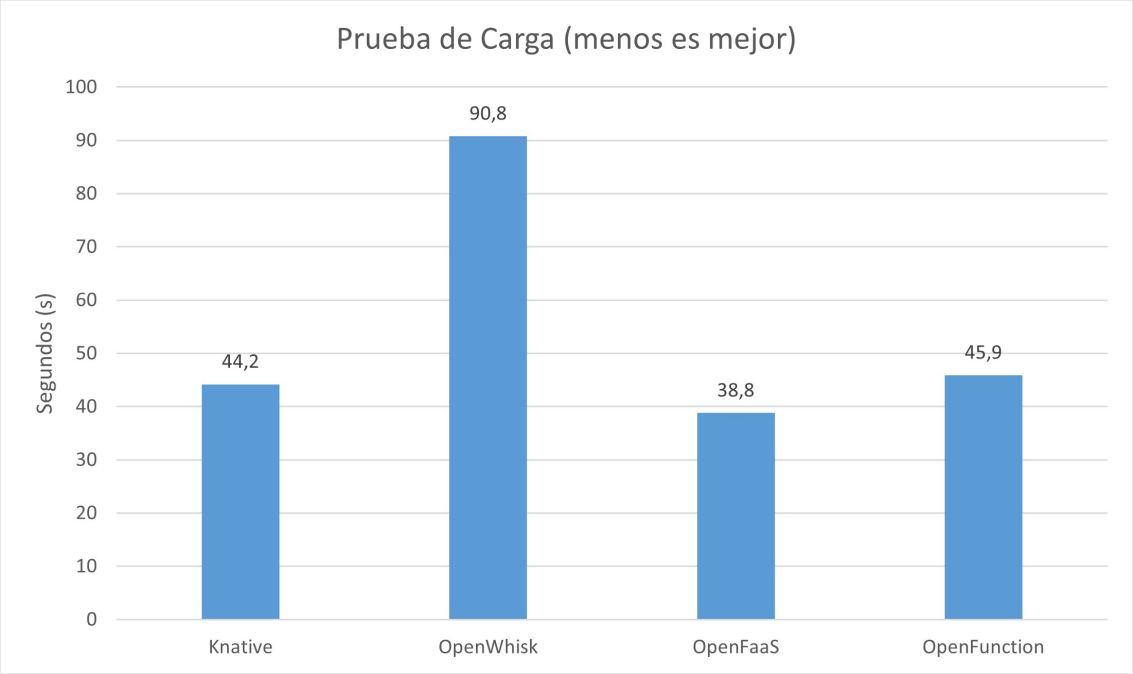

Figura 6.1 Gráfico de barras con los resultados de la prueba de carga.

si nos fijamos en OpenWhisk. Esta herramienta presenta su propio algoritmo de auto-escalado cerrado y se queda muy por detrás de las otras herramientas.

En cuanto a las diferencias entre Knative y OpenFunction. Éstas pueden ser producidas porque OpenFunction procesa las peticiones y las respuestas por si mismo y no dentro de la función como lo hace Knative. Esto ocurre aunque las dos herramientas usen Flask para esta tarea.

6.3.2 Prueba de velocidad desde cero

La prueba de velocidad desde cero implicará que herramienta es mejor en cuanto a la rapidez de crear la primera réplica. Aclarar que esta velocidad es menor que la de creación de una réplica cualquiera en medio de la prueba de carga. Aún así, estas dos velocidades son del mismo orden de magnitud.

Este experimento determinará que herramienta es la más adecuada si la función a implementar se requiere de forma muy esporádica y por ende se quieren aprovechar al máximo los recursos del servidor donde se aloje la función.

A continuación, en la Figura 6.2, se presentan los resultados de este experimento. El eje Y está presentado en segundos y tal y como dice la figura, menos tiempo implica un resultado mejor. Destacar que como se

comentó en la Subsección 6.2.3, la prueba con OpenFaaS no tiene sentido ya que, éste en la versión estándar no permite escalar desde cero. Por ende esta herramienta no será contemplada en la figura.

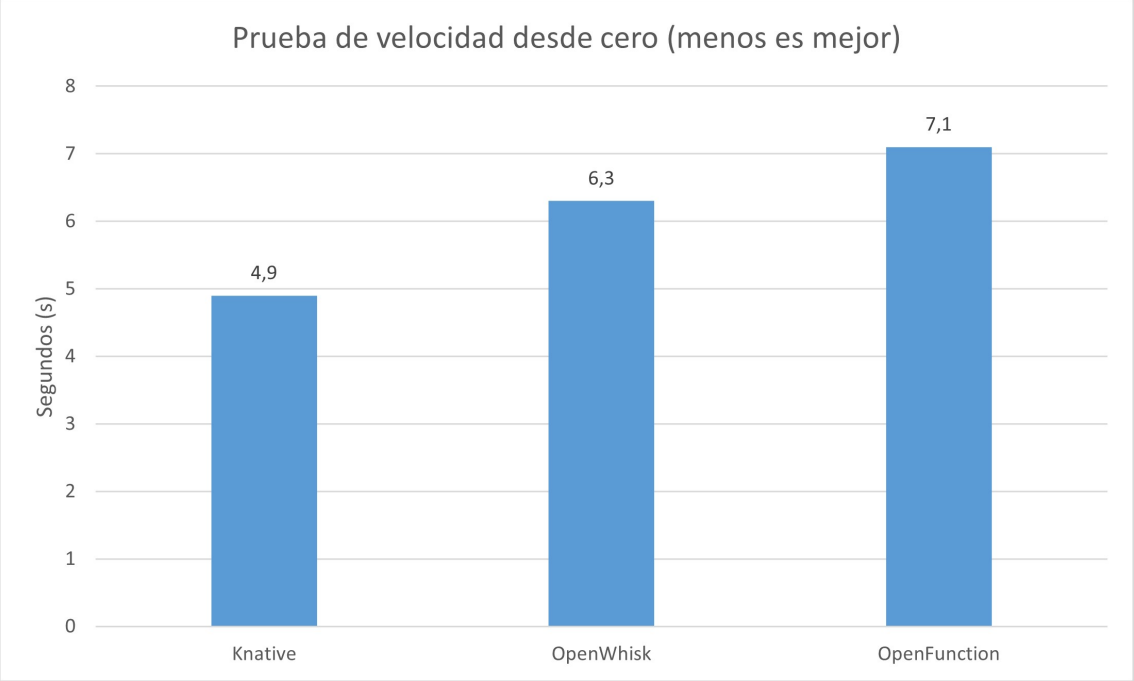

Figura 6.2 Gráfico de barras con los resultados de la prueba de velocidad desde 0.

Los resultados de este prueba son ciertamente esperados. Knative al ser la herramienta más simple es capaz de escalar la primera réplica de la forma más rápida posible. OpenWhisk sorprende en cuanto a su escalado de primera réplica, ya que es bastante lento en cuanto al escalado general en la prueba de carga. OpenFunction es más lento que Knative porque, además de lo comentado en la prueba de carga, en la primera réplica tiene que generar el *Deployment* para que Knative Serving sea capaz de lanzar los *Pods*.

6.3.3 Prueba de velocidad con función establecida

La prueba de velocidad con función establecida determinará qué herramienta es mejor en cuanto a procesamiento bruto del código implementado y en cuanto a tiempo en atender una petición única.

Este experimento determinará que herramienta es la más adecuada si las peticiones se hacen con una frecuencia tal que no sea conveniente escalar desde 0 (petición por minuto).

A continuación, en la Figura 6.3, se presentan los resultados de este experimento. El eje Y está presentado en milisegundos y tal y como dice la figura, menos tiempo implica un resultado mejor.

Los resultados de la prueba velocidad bruta son comprensibles viendo los resultados de las anteriores pruebas. Knative sale algo perdiendo porque se trabaja con imágenes Docker completas. Las imágenes utilizadas por OpenFunction y OpenFaaS son más eficientes ya que están pensadas para ser funciones. El resultado de OpenWhisk no sorprende ya que usa un formato para crear las funciones diferentes a las otras herramientas.

6.3.4 Comparativa directa entre herramientas

En está subsección se tratará de comparar de la forma más objetiva posible, es decir, mediante cálculos matemáticos, cuál herramienta es mejor de forma general, teniendo en cuenta todas las pruebas expuestas de igual manera. Posteriormente se pondrá la fórmula utilizada para obtener los resultados matemáticos.

Recordar que OpenFaaS no entrará en esta comparación por los motivos expuestos anteriormente y los expuestos en la Subsección 6.2.3.

La fórmula utilizada para calcular los pesos de calidad por prueba, en porcentaje, de cada herramienta es la siguiente:

$$\frac{100}{T_{herr} \sum \frac{1}{T_i}}$$

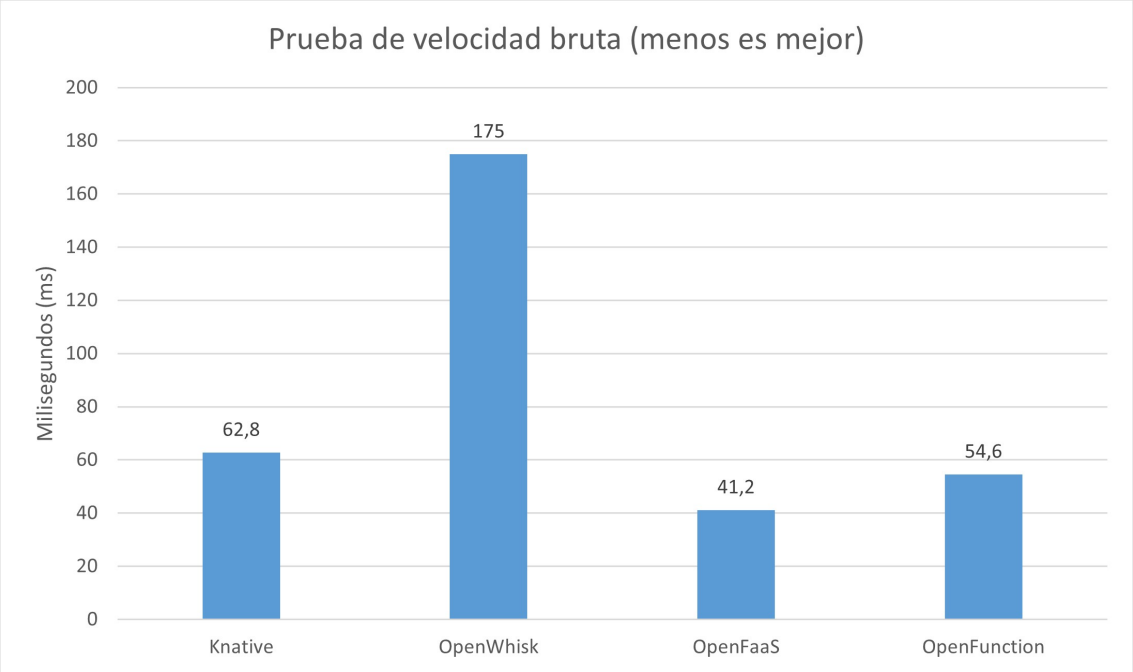

Figura 6.3 Gráfico de barras con los resultados de la prueba de velocidad bruta.

siendo T_{herr} el resultado dado en la prueba por la herramienta y T_i el resultado de cada herramienta (incluyendo la que se calcula) para ser sumado mediante el sumatorio.

El peso resultado de cada herramienta implica que esta herramienta es mejor para ese uso si el peso es mayor. La herramienta que será mejor de forma general será la que obtenga más media entre estas tres pruebas.

En la Tabla 6.1 que aparece a continuación se ven los resultados con la fórmula anterior aplicada y también con la media de los resultados indicando cuál herramienta es mejor en general.

Tabla 6.1 Comparativa objetiva global de rendimiento entre herramientas.

Herramienta	**% Prueba Carga**	**% Prueba Vel. 0**	**% Prueba Vel. Bruta**	**% Medio**
Knative	40.82	40.52	39.86	**40.40**
OpenWhisk	19.87	31.52	14.30	**21.89**
OpenFaaS	NO PROCEDE			
OpenFunction	39.31	27.96	45.84	**37.71**

Como se visualiza en la Tabla 6.1, la herramienta que más porcentaje medio ha obtenido ha sido Knative, por tanto se podría considerar cómo la mejor herramienta de forma empírica.

7 Experiencias, vivencias y evaluación sobre *Serverless Computing* para uso privado

Sólo una cosa es más dolorosa que aprender de la experiencia, y es, no aprender de la experiencia.

LAURENCE J. PETER

En este penúltimo Capítulo se detallarán las diferentes opiniones sobre las herramientas probadas, las cuales son capaces de ejemplificar el estado actual de la tecnología de *Serverless Computing* en el caso de uso con nube privada, ya que se ha trabajado con los programas más usados de esta categoría.

Se hará hincapié en los diversos problemas que se han encontrado en el tiempo de creación del trabajo, en algunas cosas a mi parecer a mejorar por las diversas herramientas, en que me ha parecido la documentación de cada una de ellas y en cómo la comparo con la nube pública con conocimiento de causa tras haber usado una plataforma de *Serverless Computing* remota cómo es AWS Lambda.

Se tratará de declarar las opiniones de forma ordenada y clara para que sean más fáciles de captar y comprender a primera vista por el lector.

7.1 Problemas encontrados durante la realización del trabajo

En esta sección se buscará exponer los distintos inconvenientes surgidos en todo el periodo de creación del trabajo de fin de grado. Esta exposición se separará por herramientas y se comentarán también los problemas encontrados no sólo en la parte práctica (instalación, puesta a punto de las herramientas y parte de auto-escalado y pruebas) sino también los surgidos en la parte de búsqueda de información y contextualización de la tecnología y de las herramientas, o cómo yo la llamo, parte teórica.

7.1.1 Problemas encontrados en la parte teórica

En general, en esta parte no ha habido demasiados problemas para completar su realización. Aún así voy a enumerar dos de ellos que al menos sí recuerdo claramente.

- **La casi total ausencia de información en español**: aunque, según mi parecer tenga un buen nivel de inglés, siempre se tarda menos tiempo en comprender ciertos conceptos complicados si éstos están en tu idioma nativo. En el único lugar donde se pueden obtener alguna información en español es en la página de *Kubernetes*.
- **Los conceptos que se suponen conocidos**: durante la realización de la especificación teórica de las diversas herramientas aparecen conceptos que son altamente necesarios para entender otros y éstos ni siquiera suelen ser introducidos mínimamente o tienen puesta una referencia a la descripción de éstos en su página oficial. Este suceso ocurre con bastante frecuencia en Knative, ya que esta herramienta es más de bajo nivel (entendiéndose como se entiende este concepto en programación) y trata de explicar más su funcionamiento desde las bases de *Kubernetes*, por ejemplo hablando de *Pods*, *Deployments* o *CRD*. En un principio, esto daba mucha sensación de pérdida de tiempo.

7.1.2 Problemas encontrados en la parte práctica

Esta parte si va a tener más contenido que la anterior, aquí se expondrán problemas que han resultado de perdidas de tiempo y de estrés bastante más grandes que los de la parte teórica. Algunos de estos problemas también han propiciado que el trabajo fuese medio abandonado en algunos momentos y que la moral en cuanto a éste fuese bajando. Las soluciones encontradas, a veces tras mucho pensar y darle vueltas, hicieron que se recuperase un poco la motivación y que se llegue a este punto del trabajo.

Knative

Knative probablemente ha sido la herramienta que menos problemas ha dado y es seguramente la que menos tiempo me haya costado poner en funcionamiento, tanto es así que esta herramienta estuvo lista ya hace practicamente 6 meses (10/7/2022). El principal problema encontrado en esta herramienta ha sido su alto consumo de hardware ya que me hizo en parte cambiar de sistema operativo y con seguridad cambiar de ordenador.

Otro problema encontrado fue el comienzo de familiarización con el uso de *Kubernetes* y *Docker* (en general, uso de la aplicación de escritorio, compresión de los ficheros *Dockerfile*) ya que esta fue la primera herramienta en la que me puse a trabajar.

Por lo demás no mucho más, si quizá me costó un cierto tiempo decidirme por qué modo tenía que usar, si *Knative Serving* o si *Knative Eventing*. Tras el recorrido del trabajo queda claro que para el uso que le he dado era evidente que *Knative Serving*, ya que es el usado para atender las peticiones HTTP.

OpenWhisk

En cuanto a OpenWhisk, probablemente una de las herramientas que más me ha decepcionado siendo además de la *Apache Foundation*. He tenido problemas variopintos que es mejor estructurar por puntos separados.

- **Problemas durante la instalación**: *kind* ha sido una herramienta que ha sido altamente útil, además de facilitar la instalación de *clusters* de *Kubernetes*, es capaz de ordenar de manera muy visual los *clusters* creados dentro de Docker Desktop. Eso sí la instalación por *kind* recomendada por OpenWhisk directamente genera un entorno en el que la herramienta no funciona (los *Pods* que sustentan OpenWhisk entran en bucle de errores fatales). Se ha necesitado de agentes externos para hacer funcionar esta plataforma. Por supuesto, esto no tiene nada que ver con *kind*, es completamente error de la herramienta de Apache.
- **Está desactualizada**: literalmente no se actualiza la versión de Python desde 2021, se sigue usando la versión 3.7 cuando acaba de salir la versión 3.11. Además no hay actualizaciones de gran contenido desde 2020. Por qué importa este punto, pues es debido a que para usar librerías no instaladas en la *runtime* de OpenWhisk no se hace cómo en otras herramientas (automáticamente), se debe hacer mediante la herramientas de entornos virtuales de Python, por ende hay que instalarse un Python compatible si se quieren instalar librerías externas (cómo es el caso de este trabajo).
- **Instalación de librerías externas**: en el anterior punto ya se comentaba algo sobre esto, la librería de OpenCV es una librería tan pesada que tras darme cuenta de lo anterior, me percaté que no cabe en una acción por defecto de OpenWhisk. Esto es un problema ya que aunque de forma teórica se puede cambiar el límite de peso de una función, la función sigue sin funcionar debido a que es una librería de por si demasiado grande y no es cargada correctamente por la herramienta. Solución final, necesité instalar otra librería de procesamiento de imágenes, Pillow y buscar información de como realizar un algoritmo parecido al que tenía con OpenCV.
- **El formato en el que llegan los binarios**: este problema me retuvo durante un tiempo, pero probablemente sea mi culpa ya que esto sí viene comentado en la documentación, aún así me parece interesante comentarlo ya que es la única herramienta que lo necesita. El contenido binario de las peticiones viene codificado en Base64 y las respuestas se deben devolver en Base64. En las demás herramientas, mediante el uso de configuración eso sí, el contenido viene directamente en un objeto *bytes* de Python.
- **Limitación de parámetros**: el más significativo es el del tamaño máximo de la petición y de la respuesta el cuál no es configurable de ninguna manera y hace que casi no se pueda usar la herramienta con la implementación que se ha realizado en este trabajo. El límite es de un 1MB, casi cualquier imagen actual es mayor que esto.
- **Durante la configuración del auto-escalado**: la herramienta simplemente no es configurable en este aspecto, viene con unos límites fijados en cuanto a concurrencia y con el escalado desde cero activado aunque las réplicas duren 10 minutos activas (cosa configurable a través de un archivo de la instalación, no es posible hacerlo con la herramienta ya instalada).

OpenFaaS

OpenFaaS es una herramienta en la que tuve menos problemas que en la anterior, pero no significa que no los tuviese y debido a la cantidad, es necesario estructurarlos mediante puntos separados.

- **Tiempo en creación de funciones**: OpenFaaS es una herramienta que se basa en Docker para crear las funciones, por ende necesita de Docker Hub para generar las imágenes y almacenarlas en algún lugar. Pues por algún motivo que desconozco, esta herramienta tarda más en subir y descargar las imágenes de Docker que si se hace de forma manual. Además esto la hace depender en exceso de la conexión a Internet que se tenga. En momentos de prueba en verano, donde dispongo de una mala conexión, se ha tardado en crear una función con OpenCV instalado hasta 30 minutos de reloj, lo cuál retrasa mucho las pruebas que se van introduciendo para ver cómo funciona la implementación realizada.
- **Diversas formas de crear una función**: en esta herramienta, hay tres formas para crear una función, la por defecto de Python, la que incluye web y la que usa Flask. Además, deberá especificarse si se usa el modo Debian o no (necesario para instalar librerías externas, en un principio perdí bastante tiempo por esto).
  Pues la única forma en que el parámetro *RAW_BODY* (necesario para enviar binarios) me ha funcionado ha sido con la versión Flask, que es una simplificación de la versión web (no se da información de la petición). Este hecho me produjo unos retrasos de bastante magnitud.
- **Funcionalidades perdidas por la versión Pro**: sin la versión Pro de OpenFaaS se pierde una interfaz que ayuda a la creación de funciones, un monitor de recursos (podría haber ayudado en la sección de pruebas) y sobretodo, que la versión estándar no incluye la posibilidad de configurar ciertas características cómo el escalado desde cero lo cuál nos saca a OpenFaaS de ser considerado completamente en las pruebas de rendimiento.

OpenFunction

OpenFunction es una herramienta que en un principio dio muchos problemas pero cuando estos problemas iniciales se solucionaron, se volvió una herramienta bastante liviana al usar *Knative Serving* como base para su funcionamiento.

El problema principal sufrido en esta herramienta ha sido que necesita de forma obligada tener conexión directa (entrada en la tabla de encaminamiento del ordenador) con la red interna del *cluster* de *Kubernetes* donde se encuentra instalada. En las otras herramientas esto no es necesario ya que disponen de un *Gateway* que se puede abrir al exterior y conectar así con el interior. Docker Desktop es una herramienta muy útil y cómoda (permite parar, reiniciar y borrar contenedores muy fácilmente) si se van a tener muchos *clusters* (cómo puede ser el caso del trabajo) divididos en contenedores independientes. El problema reside en que esta herramienta no ofrece la conexión directa con los contenedores, si es el caso de Docker Engine simple. Por tanto, OpenFunction no es compatible con Docker Desktop. Darme cuenta de esto supuso un largo tiempo y muchas pruebas, cuando al final la solución era bastante simple.

7.2 Comentarios sobre los problemas con la documentación de las herramientas

En esta sección se tratarán de destacar los inconvenientes surgidos durante la realización de la parte práctica y teórica pero centrándose en los problemas surgidos directamente por errores presentes en la documentación. El contenido de esta sección podría haberse incluido en la anterior, pero me parece más relevante separarlo, porque han sido tan importantes cómo los problemas expuestos anteriormente.

Se dividirán los problemas por herramienta para así discernir que herramienta es la mejor documentada de todas según mi experiencia.

7.2.1 Knative

Como ya comenté, Knative es la herramienta que menos problemas me ha dado. En cuanto a documentación podría decirse que para entender ciertos conceptos teóricos que se dan por sabidos podrían introducirse de alguna forma en la propia documentación y no necesariamente verlo en el sitio oficial de *Kubernetes*. Aunque está practica es muy normal a mi parecer, este dato ha sido comentado para sacarle un pequeño pero.

7.2.2 OpenWhisk

OpenWhisk es otro tema, en el caso de esta herramienta es necesario hacer un desarrollo más ordenado. A pesar de estar supervisada por Apache, la herramienta deja bastante que desear en este apartado.

- **La documentación está dividida**: la mayoría de la documentación relevante de la herramienta está en el repositorio oficial de Github o en blogs personales de los desarrolladores. Esto implica que la propia página oficial de documentación se sienta inútil y que convenga más buscar en tu buscador de confianza lo que necesites y así encontrar alguna página oficial que te resuelva el problema.
- **La instalación**: como se comentó en la anterior sección tuve que usar un blog (en este caso extraoficial) para completar la información que venía en la documentación y así conseguir que la instalación de la herramienta fuese satisfactoria. Si se siguen los pasos que se exponen en el tutorial de instalación por *kind* de la documentación algunos *Pods* no se desplegarán de forma correcta.
- **La falta de información sobre los *runtimes***: cómo ya comenté varias veces, es necesario instalar un Python específico para que las librerías externas sean instaladas. Pues para averiguar la versión que usa OpenWhisk, se necesita crear una acción que devuelva cómo respuesta la versión de Python utilizada. Esta información no se encuentra de ninguna manera en la documentación.
- **La información sobre la instalación de dependencias**: para instalar dependencias en Python la página oficial de la documentación te redirige a una web personal de James Thomas, uno de los desarrolladores de la herramienta, este blog explica de forma muy correcta cómo se realiza este proceso. El problema es que es un blog del año 2017. Este hecho simplemente ejemplifica el estado de la documentación de esta herramienta.

7.2.3 OpenFaaS

En cuanto a OpenFaaS, ha sido una herramienta menos problemática en cuanto a documentación que OpenWhisk, pero no se salva en absoluto de inconvenientes.

- **La información sobre como gestionar binarios en respuesta y petición**: la única información relativa a este tema aparece en la especificación de la etiqueta *RAW_BODY*. En la página que se habla de las diferentes formas de crear un función de tipo web no se especifica nada tampoco sobre cómo gestionar peticiones y respuestas binarias. Mediante prueba y error y mediante lo aprendido en anteriores herramientas conseguí el resultado final.
- **La información sobre la etiqueta *RAW_BODY***: la documentación no deja claro cómo hay que colocar este valor en la configuración de la función, para conseguir el resultado la he tenido que colocar mediante prueba y error.

7.2.4 OpenFunction

Finalmente en cuanto a OpenFunction, pasa algo parecido que con OpenFaaS, no he tenido demasiados inconvenientes con la documentación, aunque esto no indica en parte que la documentación sea buena. En este caso es una documentación más simple y efectiva que por ejemplo la de OpenWhisk. Problemas surgidos:

- **Información teórica muy justa**: para sacar las características teóricas de la herramienta ha habido cierto sufrimiento debido a que lo expuesto en la página es bastante pobre y no se entiende demasiado bien.
- **Incongruencias en la información**: he tenido que usar una versión antigua de la herramienta porque por ejemplo, las páginas tutoriales de creación de funciones de las últimas versiones no están actualizadas con respecto a los componentes propios de la herramienta que han ido cambiando con el paso de los lanzamientos.
- **La configuración expuesta de auto-escalado es errónea**: la configuración de ejemplo que se expone en la documentación da error al intentar crear una función con ella. Para conseguir que funcione ésta se ha tenido que usar de nuevo la prueba y error.

7.3 Comparativa nube privada contra nube pública

Tras usar AWS Lambda (y además haber documentado teóricamente otras 2 herramientas de nube remota) y las cuatro herramientas mencionadas de nube privada, me han quedado claras varias conclusiones personales que se expondrán a continuación.

- **La información disponible**: trabajando con AWS Lambda cuando he necesitado solucionar algún problema no me ha sido difícil encontrar una solución oficial rápidamente, o por el contrario, un usuario con el mismo problema ya resuelto por la comunidad. En el caso de las herramientas instalables, esto en general no ha sido así, sobretodo por la parte de usuarios con mi mismo problema.

- **La facilidad de desarrollo**: en la herramienta de Amazon hay disponible un editor en línea con el cuál puedes además probar de poco en poco los cambios hechos a tu función de manera instantánea. Esto en las herramientas instalables es más laborioso, ya que para probar un pequeño cambio, habrá que desplegar de nuevo la función con la pérdida de tiempo que eso conlleva.
- **Aprovechamiento de recursos**: un punto positivo, a mi forma de ver, de las herramientas instalables es que si se tiene potencia computacional suficiente y está en desuso, probablemente obtendrás más rendimiento que con las versiones gratuitas de las plataformas remotas que sólo ceden una potencia computacional muy pobre.
- **Compatibilidad con la nube pública**: otro punto a favor de las herramientas instalables es la portabilidad presentes en ellas al trabajar con un *cluster* de *Kubernetes* por debajo. Cualquier empresa que de servicios de computación en la nube da un servicio de *Kubernetes* en el cuál se podría instalar tus funciones de plataforma privada rápidamente si ya se quieren dejar de usar de esta forma.
- **Uso del servicio**: si se quieren implementar funciones usadas por un grupo de personas de forma privada en un contexto empresarial lo mejor será usar dentro de la empresa una plataforma privada. Esto es debido a que, suponiendo que la empresa está debidamente protegida del exterior, sólo se podrán usar internamente.
  En cambio, si se busca realizar un servicio público, lo más adecuado será usar nube remota y quitarse de problemas desde dentro de la corporación.

A pesar de ser la nube remota mucho más popular y más útil en cuanto a *Serverless Computing* (bajo mi humilde opinión), creo firmemente que la nube privada tiene un nicho de mercado aprovechable y que si quiere conseguirlo, debería mejorar ciertos temas cómo la documentación de las herramientas o que herramientas como Apache OpenWhisk no se sientan tan abandonadas.

7.4 Opinión final de cada herramienta

En esta sección final de este capítulo se comentarán que sensaciones en general me han dado las diversas herramientas trabajadas y en que las mejoraría tras haber probado las demás y tener algo de conocimiento sobre las herramientas de nube pública.

7.4.1 Knative

Knative es una herramienta más generalista, en sí es un auto-escalador de *Pods* de *Kubernetes* que trabaja con imágenes de Docker, al ser tan global, se puede usar como *Functions-as-a-Service*. A pesar de estos hechos, probablemente sea la mejor herramienta de todas ya que presenta una buena documentación y además, gracias a su simpleza en las pruebas de rendimiento sale ganadora en casi todos los aspectos.

Su mayor pero, el hecho de que la herramienta no está tan preparada para el desarrollo *FaaS* y eso puede alejar de su uso a programadores más novatos que no conozcan la estructura de *Kubernetes* y de Docker. Por lo demás, en mi opinión es la mejor herramienta.

7.4.2 OpenWhisk

OpenWhisk es probablemente la herramienta más popular de *FaaS* que da opción privada y por ello es la que más me ha decepcionado. Su entorno de desarrollo es bueno y es una herramienta medianamente rápida en el despliegue de funciones. Pero, su forma de instalar librerías externas o que no tenga posibilidad de configurar parámetros de auto-escalado (no se duda de que su algoritmo cerrado de auto-escalado sea bueno), hacen que la herramienta sea para mí la peor probablemente de este análisis.

También es de recibo recordar, las limitaciones que presentan sus funciones en cuanto a tamaño, tamaño de peticiones o respuestas ya que hacen esta herramienta prácticamente inútil para el propósito que se ha realizado en este trabajo, el procesamiento de imágenes.

7.4.3 OpenFaaS

OpenFaaS es una herramienta con un alto potencial, sobretodo si su versión Pro fuera la estándar gratuita, ya que sin ésta se limita mucho el uso del auto-escalado cómo ya se comentó en el Capítulo 6. Tanto es así, que no he podido incluir a esta herramienta en la comparación general de rendimiento (aunque si se hayan hecho las pruebas sobre ella).

En cuanto a documentación, es una herramienta mejorable siendo aún así seguramente la segunda mejor de las analizadas. Esta opinión se debe sobretodo a la información presente sobre cómo trabajar con los contenidos binarios.

Ya finalizando, y ya hablando sobre la facilidad de desarrollo que ostenta OpenFaaS, es decente, cómo mayores problemas diría que son dos: el alto tiempo de despliegue de funciones y la falta de información sobre las plantillas para crear funciones (el tema de función web, Flask o normal o si se usa Debian o no).

7.4.4 OpenFunction

OpenFunction es probablemente la herramienta menos popular de todas las analizadas (si se busca esta herramienta directamente en su buscador de confianza probablemente le salga otra cosa). Aún así, a pesar de la poca información que hay alrededor de ella, me ha sorprendido gratamente debido a su buen rendimiento (gracias a la base usada: *Knative Serving*) y su decente facilidad en cuanto al desarrollo de las funciones.

Su mayor característica es la necesidad de tener el código fuente en un repositorio de GitHub (público si se quieren ahorrar configuraciones). Mi opinión sobre esto: es una buena característica, pero también debería existir la opción de recoger el código fuente desde local por si el usuario de la herramienta prefiere esta forma.

En cuanto a la documentación de la herramienta, es objetivamente pobre, esto es debido probablemente a la poco popularidad que tiene la misma.

8 Conclusiones sobre el proyecto

Al final, no recordaremos las palabras de nuestros enemigos, pero sí el silencio de nuestros amigos.

Martin Luther King, Jr.

En este Capítulo final se tratarán de sacar diversas conclusiones mediante preguntas que en un principio no tendrán respuesta y que se tratarán de resolver con el paso de las palabras. Estas conclusiones también resumen de otra forma diferente el trabajo realizado. También se especificará que sería lo siguiente a trabajar para seguir obteniendo conocimientos sobre la tecnología de *Serverless Computing* privada.

8.1 Conclusiones

En este proyecto se ha analizado el estado de la tecnología de *Serverless Computing* tanto en el uso privado como de forma más breve en el uso público.

Se ha realizado un capítulo centrado en explicar qué es el *Serverless* y en que formas se presenta. Se incluye una pequeña comparación con las formas tradicionales de usar computación en la nube presentando beneficios y perjuicios de las diversas tecnologías. También se presentará el organismo empresarial que intenta normalizar la computación en la nube, el Cloud Native Computing Foundation (CNCF).

De forma teórica se han presentado las plataformas públicas AWS Lambda, Azure Functions y Google Cloud Functions. De estas herramientas se han destacado sus características principales de forma breve.

Se han explicado de forma más completa las herramientas instalables. También se han instalado y configurado para ser usadas. Estas plataformas son supervisadas por el CNCF. Las herramientas analizadas son las siguientes:

- **Knative** que es desarrollado por el CNCF.
- **OpenWhisk** que es desarrollado por Apache.
- **OpenFaaS** que es desarrollado por la comunidad (versión Pro desarrollada por los creadores iniciales de la herramienta).
- **OpenFunction** que es desarrollado por QingCloud.

En estas herramientas se ha creado una función de prueba básica para comprobar el funcionamiento de las mismas. También se han explicado y configurado en las herramientas los diferentes parámetros de auto-escalado más importantes.

Durante la realización de este trabajo han ido surgiendo preguntas interesantes que deberían de ser respondidas de manera bastante objetiva con los conocimientos obtenidos al considerarse terminado este proyecto. Estas respuestas definirán los conceptos estudiados y aprendidos más relevantes y además servirán como conclusión sobretodo en cuanto a la utilidad de este trabajo. Estas preguntas son las siguientes:

1. **¿Para que sirve el *Serverless Computing*?**
2. **¿Sería interesante el papel de una entidad que normalice la tecnología?**
3. **¿Qué caracteriza a las plataformas instalables?**
4. **¿Cómo de importante es el auto-escalado en *Serverless Computing*?**

5. ¿Que herramienta es objetivamente mejor tras las pruebas?

En cuanto a la primera pregunta, el *Serverless* es una tecnología cuya utilidad reside en la simplificación y el acceso más sencillo a todo el ecosistema *cloud* existente sin pasar por configuraciones que no incumben al desarrollador y que embarrarían lo más importante para un programador, el desarrollo de su aplicación.

Con la segunda pregunta hay un pequeño dilema, ya existe un organismo empresarial que se explicó brevemente este trabajo, el CNCF. Este organismo se dedica a supervisar bastantes proyectos de esta tecnología, pero viendo los resultados del trabajo, hace falta algo más para que el *Serverless Computing*, sobretodo en uso privado, mejore sustancialmente. Si se creara una organización más seria, probablemente la tecnología suba a otro nivel y explote su completo potencial.

Con respecto a la pregunta número tres, las plataformas instalables tienen todas en común el uso de *clusters* de *Kubernetes*. Este hecho simplifica bastante la gestión de los componentes de las diversas herramientas, ya que se hará todo con las utilidades de *Kubernetes*. Por tanto, la compresión de los *clusters* de esta herramienta ha sido esencial para poder sacar adelante este proyecto.

Sobre la pregunta número cuatro, el auto-escalado es una de las funciones principales que implementa esta tecnología, pero ciertamente no siempre es necesario. La necesidad del auto-escalado depende de la cantidad de peticiones que recibirá la función o *Backend* implementado. Cuanto más requerida es una aplicación más importante es esta funcionalidad. Aunque pueda parecer contradictorio, cuanto la función o *Backend* sea muy poco requerido esta funcionalidad también es altamente importante porque aparece en escena el escalado desde cero, funcionalidad que ahorrará recursos del lugar donde se tenga instalada la aplicación.

Y por último en cuanto a la quinta pregunta, ya se concluyó que conjuntando las tres pruebas realizadas, la herramienta que mejor parada salía en general era Knative desarrollada por el CNCF y cuyo objetivo último no es el de funcionar como *FaaS* sino como escalador modificado de *Kubernetes* y se adapta a *FaaS* de una manera muy precisa (los desarrolladores de la herramienta incluyen ejemplos para que funcione así).

Tras haber respondido estas preguntas clave, se comparará el tiempo usado aproximadamente en la realización de cada parte del trabajo mediante la Tabla 8.1. En esta tabla también se valorará la dificultad de cada parte de una forma subjetiva.

Tabla 8.1 Comparativa de tiempos consumidos por cada parte del proyecto.

Parte del Trabajo	% Tiempo Consumido Aprox.	Dificultad subjetiva
Capítulos teóricos	18 %	Media
Capítulo de instalación de herramientas	30 %	Media
Capítulo de creación de funciones	45 %	Difícil
Capítulo de configuración de auto-escalado y de realización de pruebas	5 %	Media
Capítulo de experiencia	1 %	Fácil
Capítulo de conclusiones	1 %	Fácil

8.2 Líneas futuras de trabajo

Para finalizar completamente el proyecto, se van a introducir cuáles serían los supuestos próximos objetivos si se quiere seguir aprendiendo sobre la tecnología *Serverless* de forma general.

- En primer lugar, lo más adecuado sería crear una herramienta basada en *Kubernetes* y personalizada para el uso más específico que se le quiera dar. Se recomienda basar las funciones en imágenes personalizadas de Docker ya que sería lo más sencillo de implementar. Por ende, la herramienta se dedicaría a gestionar el auto-escalado y en cuanto a funciones únicamente debería crear un patrón para generar los *Dockerfiles* en los que éstas se basan. Así es cómo trabajan herramientas cómo OpenFaaS o OpenFunction. Por el contrario, si se opta por crear una herramienta más parecida a OpenWhisk sería un trabajo más complejo y largo.
- En segundo lugar y si se quiere profundizar en la tecnología de forma general, lo más interesante sería realizar un trabajo parecido a este pero centrándose en las plataformas de nube pública. Así se daría otra visión más completa que la obtenida únicamente realizando este proyecto.

- En tercer y último lugar y si se desea profundizar en las plataformas que se han instalado en este trabajo, lo más útil sería acercarse a la creación de funciones que respondan a eventos y trabajar con las diferentes formas que cada herramienta presenta. Este tipo de funciones serían las que responderían a productores de eventos de bases de datos, eventos periódicos, métricas (CPU, RAM, o uso de red), etc.

Índice de Figuras

2.1 Gráfica coste-escalabilidad extraída de [4] 6

3.1 Modelo de programación de OpenWhisk extraído de [22] 20

5.1 Imagen con el fondo blanco 41
5.2 Imagen sin fondo y con canal alfa 41

6.1 Gráfico de barras con los resultados de la prueba de carga 63
6.2 Gráfico de barras con los resultados de la prueba de velocidad desde 0 64
6.3 Gráfico de barras con los resultados de la prueba de velocidad bruta 65

Índice de Tablas

2.1	Recomendaciones de uso: Serverless vs CaaS vs PaaS [2]	9
2.2	Ejemplos de uso y su porcentaje	10
3.1	Comparativa plataformas remotas	13
3.2	Comparativa de plataformas instalables	18
6.1	Comparativa objetiva global de rendimiento entre herramientas	65
8.1	Comparativa de tiempos consumidos por cada parte del proyecto	74

Índice de Códigos

4.1	install_docker.sh	24
4.2	install_kind.sh	24
4.3	install_kubectl.sh	24
4.4	install_kn.sh	25
4.5	install_Knative.sh	25
4.6	install_helm.sh	26
4.7	install_wsk.sh	27
4.8	install_wskdeploy.sh	27
4.9	kind-cluster.yaml	27
4.10	Creación del contenedor	27
4.11	mycluster.yaml	28
4.12	Comandos para lanzar la instalación en el cluster	28
4.13	Comandos de configuración básica	29
4.14	install_python3_7_11.sh	30
4.15	Comandos para añadir librerías al Runtime de Python de OpenWhisk	30
4.16	Creación de un nuevo cluster de *Kubernetes*	31
4.17	install_arkade.sh	31
4.18	install_faas-cli.sh	32
4.19	install_OpenFaaS-Chart.sh	32
4.20	Comprobación de que los contenedores asociados estén en correcto funcionamiento	32
4.21	Comandos y ficheros necesarios para configurar de forma básica OpenFaaS	33
4.22	install_docker_engine.sh	34
4.23	Creación de un nuevo cluster de *Kubernetes*	35
4.24	install_OFunctionCLI.sh	35
4.25	install_OFunction.sh	35
4.26	Comando en desuso de instalación de OpenFunction	35
4.27	Comprobación de que los contenedores de OpenFunction estén en correcto funcionamiento	36
4.28	Creación de un secret para Kubernetes	36
4.29	Configuración de acceso desde el exterior en OpenFunction	37
5.1	quitaFondo.py	40
5.2	pruebaLibreria.py	40
5.3	Peticiones mediante curl	41
5.4	Dockerfile para la implementación en Knative	42
5.5	config-app.yaml	42
5.6	kn-quitaFondo.py	42
5.7	Lanzamiento del contenedor para la app de Knative	43
5.8	Imposición de la configuración (creación de objetos)	44
5.9	Comandos útiles para comprobar si Knative se inició correctamente	44
5.10	Comprobaciones lanzamiento de la implementación en Knative	44
5.11	Recogida de URL del servicio de Knative	45

5.12 Petición curl Knative 45
5.13 __main__.py 45
5.14 Pasos para la creación de una acción en OpenWhisk 47
5.15 Comando para obtención de URL de acción en OpenWhisk 47
5.16 Comandos para comprobar errores de acciones en tiempo de ejecución 47
5.17 Petición curl OpenWhisk 48
5.18 Comando para crear el esquema de ficheros 48
5.19 quitafondo.yml 48
5.20 handler.py 49
5.21 requirements.txt 49
5.22 Comandos para lanzar en un nuevo la función a partir de su fichero YAML 49
5.23 Comandos para comprobar el despliegue de una función en OpenFaaS 50
5.24 Petición curl OpenFaaS 50
5.25 quitafondo.yaml 51
5.26 main.py 52
5.27 requirements.txt 52
5.28 Comandos para crear, eliminar y comprobar la creación de funciones en OpenFunction 52
5.29 Comandos para obtener la URL en OpenFunction 53
5.30 Petición curl OpenFunction 53

6.1 prueba_carga.sh 56
6.2 prueba_vel0.sh 57
6.3 prueba_vel_bruta.sh 57
6.4 prueba_total.sh 58
6.5 config-map.yaml 60
6.6 config-app.yaml 60
6.7 quitafondo.yml 61
6.8 quitafondo.yaml 62

Bibliografía

[1] "Wikipedia EN Cloud Computing," Fecha de acceso: 19/03/2022. [Online]. Available: https://en.wikipedia.org/wiki/Cloud_computing

[2] CNCF, "CNCF Serverless Paper," Fecha de acceso: 04/02/2022. [Online]. Available: https://www.cncf.io/.

[3] "Serverless Guide by GitHub Community," Fecha de acceso: 30/03/2022. [Online]. Available: https://serverless.github.io/guide/

[4] K. Chowhan, *Hands-on serverless computing : build, run, and orchestrate serverless applications using AWS Lambda, Microsoft Azure Functions, and Google Cloud Functions*, 2018.

[5] "CaaS RedHat," Fecha de acceso: 13/06/2022. [Online]. Available: https://www.redhat.com/en/topics/cloud-computing/what-is-caas

[6] "PaaS Wikipedia EN," Fecha de acceso: 13/06/2022. [Online]. Available: https://en.wikipedia.org/wiki/Platform_as_a_service

[7] "MBaaS Wikipedia EN," Fecha de acceso: 15/06/2022. [Online]. Available: https://en.wikipedia.org/wiki/Mobile_backend_as_a_service

[8] "Frontend and backend Wikipedia EN," Fecha de acceso: 15/06/2022. [Online]. Available: https://en.wikipedia.org/wiki/Frontend_and_backend

[9] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud Programming Simplified: A Berkeley View on Serverless Computing," 2 2019. [Online]. Available: http://arxiv.org/abs/1902.03383

[10] A. Passwater, "2018 Serverless Community Survey: huge growth in serverless usage," 7 2018, Fecha de acceso: 15/06/2022. [Online]. Available: https://www.serverless.com/blog/2018-serverless-community-survey-huge-growth-usage/

[11] "CNCF Official Web Page," Fecha de acceso: 16/06/2022. [Online]. Available: {https://www.cncf.io/}, {https://github.com/cncf/foundation/}

[12] "Wikipedia EN Serverless Computing," Fecha de acceso: 01/03/2022. [Online]. Available: https://en.wikipedia.org/wiki/Serverless_computing

[13] "Serverless Market Share," 2018, Fecha de acceso: 27/04/2022. [Online]. Available: {https://thenewstack.io/this-week-in-numbers-serverless-now-rivals-vms-delivered-as-a-service/},{https://www.datadoghq.com/state-of-serverless/}

[14] AWS, "AWS Lambda Guía para desarrolladores," Fecha de acceso: 27/04/2022. [Online]. Available: https://docs.aws.amazon.com/lambda/index.html

[15] Microsoft, “Azure Functions Docs,” Fecha de acceso: 25/04/2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/

[16] Google, “Google Cloud Functions,” Fecha de acceso: 25/04/2022. [Online]. Available: https://cloud.google.com/functions/docs/2nd-gen

[17] “AWS Lambda Custom Runtime,” Fecha de acceso: 01/06/2022. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/runtimes-custom.html

[18] “AWS Lambda Runtimes,” Fecha de acceso: 01/06/2022. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-concepts.html#gettingstarted-concepts-runtime

[19] “Kubernetes Docs,” Fecha de acceso: 02/06/2022. [Online]. Available: https://kubernetes.io/docs/home/

[20] CNCF, “Keda Docs,” Fecha de acceso: 08/04/2022. [Online]. Available: {https://keda.sh/docs/2.6/}, {https://github.com/kedacore/keda-docs}

[21] ——, “Knative Docs,” Fecha de acceso: 08/04/2022. [Online]. Available: {https://knative.dev/docs/}, {https://github.com/knative/docs}

[22] Apache, “OpenWhisk Docs,” Fecha de acceso: 08/04/2022. [Online]. Available: {https://openwhisk.apache.org/documentation.html},{https://github.com/apache/openwhisk}

[23] OpenFaaS, “OpenFaaS Docs,” Fecha de acceso: 08/04/2022. [Online]. Available: {https://docs.openfaas.com/},{https://github.com/openfaas/faas}

[24] QingCloud, “OpenFunctions Docs,” Fecha de acceso: 08/04/2022. [Online]. Available: {https://openfunction.dev/docs/},{https://github.com/OpenFunction/OpenFunction}

[25] “Kind Docs,” Fecha de acceso: 22/06/2022. [Online]. Available: https://kind.sigs.k8s.io/

[26] J. Thomas, “Ayuda instalación paquetes Python en OpenWhisk,” Fecha de acceso: 12/07/2022. [Online]. Available: https://jamesthom.as/2017/04/python-packages-in-openwhisk/

[27] “OpenCVDocs for Python,” Fecha de acceso: 06/07/2022. [Online]. Available: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

[28] “Pillow Docs,” Fecha de acceso: 21/10/2022. [Online]. Available: https://pillow.readthedocs.io/en/stable/index.html