Metodología para la recolección y curación de un dataset de aplicaciones serverless (desarrolladas con Serverless Framework y AWS Lambda) de código abierto en GitHub para su caracterización

Ángel Camilo Chávez Moreno

Tutora:
Cristina Lucía Abad Robalino, PhD.

Una tesis submitida para el grado de:
Magister en Ciencias de la Computación

Dedicatoria

Dedico este trabajo a Dios, cuya fuerza ha sido el principal motor para llegar a esta milla.

A mi madre, Ingrid Solange Moreno Flores, por empujar fervientemente mi preparación académica, especialmente en el aprendizaje del idioma inglés, y por su soporte extraordinario para dirigirme con propósito en mi vida.

A mi hermano, Diego Alejandro Chávez Moreno, y a Fátima Gabriela Serrano Chérrez, quienes junto a mi madre representan día a día lo más preciado que tengo: mi familia.

A la memoria de mi padre, Ángel Napoleón Chávez López; de mis abuelos, Rosa Esther Flores Pérez, mi Mami Rosa, y Carlos Enrique Moreno Solís, mi Tata, de quienes sigo aprendiendo y reflexionando hasta el día de hoy.

A mis amados grupos de amigos: Grupillo, DxP y Cool Lions, por la ejemplar camaradería y amistad que compartieron conmigo en los días más complicados.

Y a vos, Ximena Lissette Larrea Espinoza. Tu amor y apoyo incondicional durante todos estos meses de trabajo han sido una aventura por sí misma. Eres un verdadero ejemplo de resiliencia y empatía; no tienes más que mi absoluta gratitud por siempre estar a mi lado.

Agradecimientos

Agradezco profundamente a mi tutora de tesis, Cristina Lucía Abad Robalino, por su excepcional guía, su paciencia y por brindarme la oportunidad de trabajar bajo su dirección. Sus estándares de excelencia fueron fundamentales para elevar la calidad de esta investigación a un nivel que simboliza con fidelidad lo que representa ser parte de esta prestigiosa institución. Gracias a ello, en el 2025, pude participar en la IEEE International Conference on Cloud Engineering (IC2E) en Rennes, Francia, lo cual constituye un capítulo trascendental en mi formación académica.

Siguiendo lo anterior, también quiero agradecer a Guillaume Pierre y a todo el comité evaluador que aceptó este trabajo para su publicación.

Al coordinador del programa de maestría, Daniel Erick Ochoa Donoso, por facilitar las gestiones necesarias para la consecución de este grado.

Asimismo, agradezco de manera especial a Juan José García Cedeño, mi hermano de vida, cuyo feedback técnico y revisión crítica del trabajo final aportaron una perspectiva esencial para el cierre de este proyecto y su eventual presentación.

Evaluadores

Firmado electrónicamente por:
CRISTINA LUCIA ABAD ROBALINO
Validar únicamente con FirmaEC

Cristina Lucía Abad Robalino, PhD.
Tutora
Escuela Superior Politécnica del Litoral

Firmado electrónicamente por:
CARLOS JOSEPH MERA GOMEZ
Validar únicamente con FirmaEC

Carlos Joseph Mera Gómez, PhD.
Evaluador
Escuela Superior Politécnica del Litoral

Declaración expresa

Yo, Ángel Camilo Chávez Moreno, acuerdo y reconozco que:

La titularidad de los derechos patrimoniales de autor (derechos de autor) del proyecto de graduación corresponderá al autor o autores, sin perjuicio de lo cual la ESPOL recibe en este acto una licencia gratuita de plazo indefinido para el uso no comercial y comercial de la obra con facultad de sublicenciar, incluyendo la autorización para su divulgación, así como para la creación y uso de obras derivadas. En el caso de usos comerciales se respetará el porcentaje de participación en beneficios que corresponda a favor del autor o autores.

La titularidad total y exclusiva sobre los derechos patrimoniales de patente de invención, modelo de utilidad, diseño industrial, secreto industrial, software o información no divulgada que corresponda o pueda corresponder respecto de cualquier investigación, desarrollo tecnológico o invención realizada por mí durante el desarrollo del proyecto de graduación, pertenecerán de forma total, exclusiva e indivisible a la ESPOL, sin perjuicio del porcentaje que me corresponda de los beneficios económicos que la ESPOL reciba por la explotación de mi innovación, de ser el caso.

En los casos donde la Oficina de Transferencia de Resultados de Investigación (OTRI) de la ESPOL comunique al autor que existe una innovación potencialmente patentable sobre los resultados del proyecto de graduación, no se realizará publicación o divulgación alguna, sin la autorización expresa y previa de la ESPOL.

Guayaquil, 26 de enero del 2026

Firmado electrónicamente por:
ANGEL CAMILO CHAVEZ MORENO
Validar únicamente con FirmaEC

Ángel Camilo Chávez Moreno

Resumen

Funcion como servicio, o Function-as-a-Service (FaaS) es la base de la computación *serverless*, permitiendo a los desarrolladores implementar aplicaciones fácilmente sin administrar recursos informáticos. Con un enfoque de Infraestructura como Código (IaC), *frameworks* como Serverless Framework utilizan configuraciones YAML para definir e implementar APIs, tareas, flujos de trabajo y aplicaciones basadas en eventos en proveedores de la nube, lo que promueve un desarrollo sin fricciones. Como ocurre con cualquier ecosistema en rápida evolución, se necesitan conocimientos actualizados sobre cómo se utilizan estas herramientas en proyectos reales. Basándose en la metodología establecida por el *dataset* Wonderless para computación *serverless* (y aplicando múltiples pasos de filtrado nuevos), OpenLambdaVerse aborda esta deficiencia mediante la creación de un conjunto de datos de repositorios actuales de GitHub que utilizan Serverless Framework en aplicaciones que contienen una o más funciones de AWS Lambda. Posteriormente, analizamos y caracterizamos este conjunto de datos para comprender el estado del arte en arquitecturas sin servidor basadas en este *stack*. A través de este análisis, obtenemos información importante sobre el tamaño y la complejidad de las aplicaciones actuales, los lenguajes y entornos de ejecución que emplean, cómo se activan las funciones, la madurez de los proyectos y sus prácticas de seguridad (o su ausencia). OpenLambdaVerse ofrece así un recurso valioso y actualizado tanto para profesionales como para investigadores que buscan comprender mejor la evolución de las cargas de trabajo sin servidor.

Palabras Clave: *serverless*, computación *serverless*, función como servicio, computación en la nube, caracterización, minería de repositorios.

Abstract

Function-as-a-Service (FaaS) is at the core of serverless computing, enabling developers to easily deploy applications without managing computing resources. With an Infrastructure-as- Code (IaC) approach, frameworks like the Serverless Framework use YAML configurations to define and deploy APIs, tasks, workflows, and event-driven applications on cloud providers, promoting zero-friction development. As with any rapidly evolving ecosystem, there is a need for updated insights into how these tools are used in real-world projects. Building on the methodology established by the Wonderless dataset for serverless computing (and applying multiple new filtering steps), OpenLambdaVerse addresses this gap by creating a dataset of current GitHub repositories that use the Serverless Framework in applications that contain one or more AWS Lambda functions. We then analyze and characterize this dataset to get an understanding of the state-of-the-art in serverless architectures based on this stack. Through this analysis we gain important insights on the size and complexity of current applications, which languages and runtimes they employ, how are the functions triggered, the maturity of the projects, and their security practices (or lack of). OpenLambdaVerse thus offers a valuable, up-to-date resource for both practitioners and researchers that seek to better understand evolving serverless workloads.

Index Terms: serverless, serverless computing, function-as-aservice, cloud computing, characterization, repository mining.

Índice general

Resumen vi

Abstract vii

Abreviaturas x

Índice de figuras xii

Índice de cuadros xiii

Lista de fragmentos de código xiv

1 Introducción 1

2 Recolección y curación de dataset 5
2.1 Búsqueda de archivos de configuración de Serverless Framework . . . 5
2.1.1 Notas sobre el consumo de la API REST de GitHub . . . 5
2.2 Filtrado de URLs . . . 6
2.3 Obtención de metadatos del repositorio . . . 6
2.4 Pasos de filtrado adicionales . . . 7
2.5 Selección de proyectos con AWS como plataforma de destino . . . 7
2.6 Clonación . . . 7

3 Caracterización del dataset 8
3.1 Año de creación . . . 8
3.2 *Plugins* . . . 8
3.3 Complejidad de código fuente (medida a través de líneas de código, o LOC) . . . 9
3.4 Bytes de código . . . 10
3.5 Entornos de ejecución . . . 10
3.6 Tópicos . . . 11
3.7 Tamaño de repositorios . . . 11
3.8 Contribuciones en los repositorios . . . 11
3.9 *Event triggers* . . . 12
3.10 Metadatos de GitHub relacionados a la seguridad del repositorio . . . 13

4 Comparación con trabajo previo 17
4.1 Entornos de ejecución . . . 18
4.2 *Plugins* . . . 18

4.3 Complejidad (LOC) 19
4.4 *Event triggers* 19

5 Discusión **20**

6 Amenazas a la validez **22**
6.1 Validez externa 22
6.1.1 Alcance y representatividad del dataset 22
6.1.2 *Stack* elegido (Serverless Framework + AWS Lambda) 22
6.2 Validez interna 23
6.2.1 Criterio y limitaciones del filtrado de repositorios 23
6.2.2 Interpretación de prácticas de seguridad a partir del análisis de los metadatos 23

7 Conclusiones **24**

Referencias **25**

Archivos adjuntos **28**
Anexo A: Código fuente del script desarrollado para la búsqueda de repositorios en GitHub . . . 29
Anexo B: Aplicaciones de muestra del dataset 36

Abreviaturas

API Application Programming Interface.

CVS Commercial Vehicle Services.

DVSA Driver and Vehicle Standards Agency.

FaaS Functions as a Service.

HTTP Hypertext Transfer Protocol.

IaC Infrastructure as Code.

LOC Lines of Code.

MSR Mining Software Repositories.

S3 Amazon Simple Storage Service.

SNS Amazon Simple Notification Service.

SQS Amazon Simple Queue Service.

Simbología

Índice de figuras

2.1 Metodología de extracción de repositorios. 6

3.1 Años de creación de los proyectos en *OpenLambdaVerse* (fecha de corte: 24 de abril del 2025). 9
3.2 Distribución acumulada (escala logarítmica) de las LOC para todo el código en los repositorios (no sólo los *handlers* de funciones). 12
3.3 Distribución acumulada (escala logarítmica) del tamaño de los repositorios. 15
3.4 Distribución acumulada del número de *event triggers* por repositorio. 15
3.5 Número de event *triggers* vs. LOC 16

5.1 Arquitectura IoT de ejemplo del dataset de Cloudscape [31]: TechConnect (deportes impulsados por tecnología con aprendizaje automático de IoT). 21

1 Vista previa de la aplicación. 36
2 Vista previa de la aplicación. 37

Índice de cuadros

3.1 Plugins de Serverless Framework más utilizados en *OpenLambaVerse*. 10
3.2 Líneas de código fuente por lenguaje de programación. 11
3.3 Lenguajes de programación más relevantes por tamano de código fuente (en bytes). 13
3.4 Entornos de ejecución con mayor presencia en *OpenLambdaVerse*. 13
3.5 Tópicos con mayor presencia en el *dataset*. 14
3.6 *Event triggers* utilizados para llamar a las funciones *serverless*. 14

4.1 Datasets de aplicaciones *serverless* y caracterizaciones realizadas. 17

Lista de fragmentos de código

1 src/get_urls.sh 35

1

Introducción

Desde el lanzamiento de AWS Lambda en 2014, la adopción del modelo de *Function-as-a-Service* (FaaS) ha aumentado considerablemente. Las soluciones FaaS (*serverless*) proporcionan las herramientas necesarias para delegar la gestión de la infraestructura del desarrollador al proveedor, al mismo tiempo que ofrecen un atractivo modelo de costos acorde al uso y el aprovisionamiento dinámico de recursos según la demanda [1].

Proveedores de servicios en la nube como Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), Oracle Cloud Infrastructure (OCI), así como alternativas de código abierto como OpenWhisk y Kubeless permiten la implementación de arquitecturas complejas basadas en eventos, con funcionalidades tales como la gestión de estados, la orquestación de flujos de trabajo y la integración con servicios en la nube como bases de datos, sistemas de almacenamiento, colas de mensajes, flujos de trabajo con IA/ML, IoT y *backends* móviles. También existen herramientas que agilizan el desarrollo y la implementación de aplicaciones *serverless*, como el popular Serverless Framework [2].

A pesar de estas y otras ventajas, la computación *serverless* sigue presentando desafíos que impulsan diversos tópicos de investigación, tales como la mitigación de la latencia de arranque en frío, la optimización del rendimiento y la rentabilidad para diversas cargas de trabajo, la solución de problemas de seguridad y la habilitación de la interoperabilidad.

Para asistir a la investigación y el desarrollo, se han recopilado y caracterizado aplicaciones que emplean funciones *serverless* en estudios previos [3–6]. El *dataset* Wonderless [3], [7], [8] consta de 1877 proyectos de GitHub que utilizan Serverless Framework. El artículo también incluye una breve caracterización del *dataset*, analizando los lenguajes de programación, entornos de ejecución y contribuciones en los repositorios. *The Open-Source Serverless Search* (OS^3) [5] extendió la recolección de proyectos con un enfoque mejorado, y su código fuente para la recopilación/curación de datos excluye repositorios sin licencia o duplicados. Esto resultó en un nuevo *dataset* [9] de 5981 aplicaciones. OS^3 se centra en mejorar la búsqueda de código, pero no incluye una caracterización de los datos. AWSomePy [6, 10] también amplió la metodología introducida en Wonderless para obtener un *dataset* con 145 aplicaciones *serverless* implementadas en Python utilizando Serverless Framework para AWS Lambda.

La caracterización incluyó el análisis de *plugins*, la complejidad del código fuente y el uso de servicios en la nube y API.

Con cada nuevo esfuerzo de caracterización, la comunidad comprende mejor cómo se desarrollan las aplicaciones *serverless*, y este conocimiento permite afrontar varios de los mayores desafíos en la recolección automática y el análisis de aplicaciones *serverless* de código abierto.

El primero de estos desafíos es la selección de una fuente apropiada de repositorios de código fuente. Trabajos previos, tales como el estudio realizado por Eismann et al. en 2022, recopilaron aplicaciones *serverless* de una amplia gama de fuentes, incluyendo proyectos públicos en GitHub, literatura académica e industrial, y el área de la computación científica. A partir de este sólido universo de fuentes, se obtuvo un *dataset* representativo para su posterior análisis. Otros estudios, como "The Wonderless Dataset for Serverless Computing" de Eskandani y Salvaneschi en 2020, también consideraron la posibilidad de utilizar fuentes adicionales de proyectos públicos, tales como el Repositorio de Aplicaciones sin Servidor de Amazon Web Services (AWS SAR). Sin embargo, esta fuente tiene que ser usada en conjunto con otras previamente mencionadas, puesto que AWS SAR se limita únicamente a implementaciones proveídas por AWS, lo que podría afectar la representatividad general del *dataset* que vaya a obtenerse. Otras plataformas de software tienen el potencial de proporcionar proyectos valiosos adicionales, incluyendo GitLab, Bitbucket y otras.

A medida que más estudios avanzan hacia la automatización de esta tarea inicial de recopilación, GitHub se ha posicionado como la fuente principal de proyectos públicos. Esta proporciona una gran cantidad de repositorios, con alrededor de 294 millones de proyectos públicos según la última consulta de búsqueda [11], y alberga una enorme comunidad de desarrolladores y colaboradores, con alrededor de 257 millones de usuarios [12]. Este gran volumen de código y colaboración refuerza los argumentos a favor de la representación de cualquier *dataset* derivado de una búsqueda de aplicaciones *serverless*, utilizando criterios específicos. Esta fuente resulta especialmente útil para comprender el estado actual de la adopción y las prácticas en el ecosistema *serverless* de código abierto. GitHub también proporciona su API para realizar consultas en sus repositorios, recopilar metadatos de interés y utilizar criterios de búsqueda avanzados. Esto es ideal para los intentos de *scraping* automatizado, por lo que un nuevo *dataset* utilizaría GitHub como fuente principal de proyectos.

Otra dificultad se basa en las implementaciones no estandarizadas de las aplicaciones *serverless*. Se sabe que el ecosistema actual de las aplicaciones *serverless* es diverso, con una cantidad considerable de *frameworks* y plataformas disponibles para el desarrollo y la implementación. Para comenzar con cualquier intento de recopilación, es importante definir los criterios para que un repositorio se considere una aplicación *serverless* en primer lugar. Si se consideran los principios básicos de FaaS, es comprensible que las aplicaciones *serverless* posean una fuerte correlación entre el código y la infraestructura. Los archivos de Infraestructura como Código (IaC), como por ejemplo *serverless.yml* (utilizado por Serverless Framework) o *template.yaml* (utilizado por AWS), definen el comportamiento y las dependencias que utilizarán las funciones implementadas en el código. Se puede usar la información de estas plantillas para identificar funciones, *triggers* de eventos y entornos de ejecución, que forman parte de una visión elevada de la arquitectura de las aplicaciones *serverless*. Dado que las plantillas en sí mismas tampoco son estándar, también es difícil conciliar todos los archivos disponibles públicamente para un *dataset* lo suficientemente representativo. Se han realizado esfuerzos en este sentido, como en el estudio OS^3 de 2023 para mejorar la búsqueda automatizada de código actual en GitHub. Este estudio incluye otros archivos de plantilla utilizados por Azure e IBM, así como archivos de plantilla específicos de AWS. Para

ofrecer una progresión comprensible, se siguen las ideas de investigaciones previas para utilizar las plantillas propias de Serverless Framework. Por lo tanto, el esfuerzo de este trabajo debe centrarse en obtener la mayor cantidad posible de estas referencias a *serverless.yml* a partir de la búsqueda de código inicial.

Es importante considerar que, a pesar de delimitar los criterios iniciales para la selección de repositorios, una parte significativa del conjunto de datos podría consistir en prototipos rápidos, laboratorios, proyectos de prueba y otros trabajos inconclusos. Por lo tanto, es necesario identificar aquellos metadatos que describan de forma consistente el propósito de la aplicación, y evaluarlos a través de heurísticas respaldadas por investigación previa. La comparativa podría incluir datos como la estructura de los directorios y subdirectorios de los proyectos, archivos `README.md`, descripciones, historial de *commits* e, incluso, información acerca de los propietarios de estos proyectos.

Recopilar repositorios de GitHub conlleva otro desafío que debe considerarse: los límites de la API. Cualquier herramienta de *scraping* debe considerar los límites de la API de la fuente. En este caso, debe incluir funcionalidad que cumpla con los límites de consumo de la API REST de GitHub (en su versión más reciente, al 28 de noviembre del 2022, y con cambios en la búsqueda de código introducidos en el año 2023).

Finalmente, se presenta un desafío adicional sobre el uso de licencias en proyectos de código abierto. Las licencias permisivas, como la licencia MIT y otras como Apache 2.0 y BSD, indican explícitamente que el código de estas aplicaciones *serverless* puede usarse en software propietario (de código cerrado), modificarse y distribuirse, siempre que se cumplan las pocas condiciones explícitas.

Para atender los desafíos previamente mencionados, se presenta *OpenLambdaVerse*. Este trabajo busca actualizar y ampliar el *dataset* de Wonderless a través de la revisión y extensión de la metodología existente. Esta mejora se basará en trabajo previo, y en las mejores prácticas de la comunidad de Minería de Repositorios de Software (MSR, por sus siglas en inglés: Mining Software Repositories). También se incluirá la actualización del código fuente proporcionado para la búsqueda y recolección de repositorios, asegurando compatibilidad con la versión más reciente de la API REST de GitHub. Esto permitirá cumplir con las políticas de uso y los límites más recientes para la búsqueda de código y la recopilación de metadatos. Con esta extensión, tanto el nuevo *dataset* como la metodología mejorada servirán como referencia valiosa para desarrolladores e investigadores en el campo.

Las contribuciones clave de este trabajo incluyen:

1. Un *dataset* actualizado y público de aplicaciones con licencia en GitHub que utilizan Serverless Framework para AWS [1]. Este conjunto de datos aborda los desafíos de la selección de fuentes y las implementaciones no estándar, centrándose en un subconjunto específico y bien definido de proyectos.
2. Una nueva herramienta de extracción de repositorios que cumple con los límites de uso de la API REST de GitHub [2]. Se basa en la metodología Wonderless y en la implementación de las mejores prácticas de la comunidad de MSR. Esto garantiza una recopilación de proyectos eficiente y conforme a las normativas de la API.
3. Un análisis del *dataset* que revela tendencias actuales en el desarrollo de aplicaciones *serverless* de código abierto. Estos hallazgos fueron comparados con los resultados de investigaciones previas,

[1] https://zenodo.org/records/16533581
[2] https://github.com/disel-espol/openlambdaverse

aportando nuevos *insights* sobre la evolución de este ecosistema.

4. Los artefactos de este trabajo fueron difundidos y validados ante la comunidad científica mediante la publicación y presentación oral del *paper* titulado *"OpenLambdaVerse: A Dataset and Analysis of Open-Source Serverless Applications"* en la conferencia *IEEE International Conference on Cloud Engineering* (IC2E 2025), celebrada en Rennes, Francia [13].

2

Recolección y curación de dataset

OpenLambdaVerse se construyó mediante una metodología revisada que mejora significativamente el proceso inicial de extracción de código fuente. Esta metodología realiza una extensión de la implementación de Wonderless, incorporando mejoras clave introducidas en investigaciones previas, particularmente de OS^3 y AWSomePy. Además, se refinaron las heurísticas existentes y se agregaron nuevos criterios derivados de las mejores prácticas de la comunidad de investigación de MSR, lo que permite una recopilación de proyectos más rigurosa y específica.

Los pasos más relevantes de esta nueva metodología se presentan en la Figura 2.1, y cada fase del proceso de recolección se describe a continuación:

2.1 Búsqueda de archivos de configuración de Serverless Framework

Siguiendo la metodología establecida de Wonderless y AWSomePy, el proceso inició con una búsqueda exhaustiva de código mediante la API REST de GitHub. Esta búsqueda se realizó el 24 de abril de 2025. Se buscaron repositorios que contengan al menos un archivo de configuración *serverless.yml*. Este primer paso recuperó un total de 34,320 URLs de archivos de configuración.

2.1.1 Notas sobre el consumo de la API REST de GitHub

El *script* desarrollado (1) para el uso del *endpoint* de búsqueda de código (*code search*) considera los siguientes criterios técnicos para garantizar el cumplimiento de las normativas de GitHub:

- **Límites de tamaño:** Solo se indexan archivos menores a 384 KB. Dado que los archivos YAML de gran escala son inusuales (solo se identificaron 3 entre 86 KB y 168 KB), esta restricción no compromete la integridad del *dataset*. A diferencia de *Wonderless*, no se excluyeron archivos menores a 0.5 KB en esta fase.

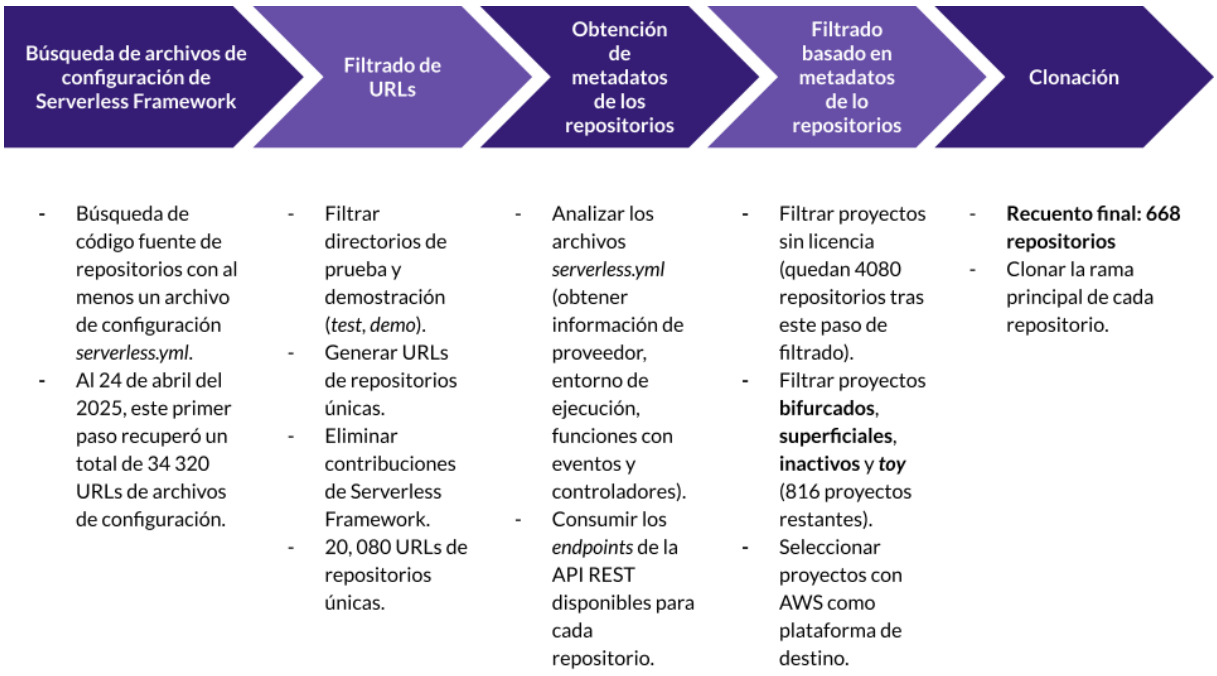

Figura 2.1: Metodología de extracción de repositorios.

- **Gestión de solicitudes:** Para no exceder el límite de 5000 solicitudes por hora de la API, se implementó un temporizador de suspensión (*sleep timer*).
- **Restricciones de búsqueda:** El *endpoint* de búsqueda limita el tráfico a 10 solicitudes por minuto. El *script* asegura el cumplimiento mediante esperas controladas de 7 segundos entre consultas.

2.2 Filtrado de URLs

Se descartaron las URLs que contenían las palabras clave *example*, *test* y *demo* en su estructura de directorios, conversando este mismo criterio establecido en investigaciones anteriores.

2.3 Obtención de metadatos del repositorio

Se extrajeron los metadatos de cada repositorio analizando el archivo *serverless.yml* y utilizando los *endpoints* para repositorios de la API REST de GitHub, con la finalidad de obtener la mayor cantidad posible de metadatos. Esto incluye: *plugins*, entornos de ejecución, eventos, proveedor, tamaño, bifurcaciones (*forks*), estrellas, temas (*topics*), lenguaje de programación principal, indicadores: [archivado, deshabilitado, (es una) bifurcación], bytes por lenguaje de programación, colaboradores, informes privados de vulnerabilidades, etiquetas, fecha de la última confirmación (*commit*), observadores, incidencias abiertas y licencia. Estos datos se compilan en un archivo de salida JSONL, donde cada objeto

JSON contiene los pares clave-valor de los metadatos de este repositorio.

2.4 Pasos de filtrado adicionales

Al igual que en OS[3], se excluyeron los proyectos sin licencia. Un repositorio sin licencia no se puede reutilizar sin el permiso del propietario. Esto permite que el *scraping* siga una ruta más directa hacia aplicaciones creadas deliberadamente como código abierto, lo que deriva en 4,080 repositorios.

Antes de clonar los repositorios, se filtraron proyectos adicionales a partir de los siguientes criterios recopilados en la revisión de artículos en la comunidad de investigación de MSR:

- **Bifurcaciones (*forks*).** Dado que las bifurcaciones se derivan de un proyecto existente, contienen código duplicado y distorsionan los resultados [14].
- **Proyectos superficiales.** Se definen los proyectos superficiales como aquellos con un tamaño inferior a 100 KB, una heurística para filtrar proyectos triviales o de juguete [14].
- **Proyectos inactivos.** Los proyectos inactivos se definen como aquellos que no se han actualizado en los últimos 24 meses; se filtran porque pueden no representar las tendencias recientes en el desarrollo de aplicaciones [15–19]. Este criterio reemplaza el umbral de madurez del proyecto Wonderless (último-primer commit ≥ 1 año), que podría derivar del propósito de estos proyectos; por ejemplo, creados como parte de una competición. Sin embargo, la mayoría de los proyectos creados en esos contextos se filtran según los criterios establecidos hasta el momento (p. ej., tamaño), y se decidió no excluir los proyectos recientes, ya que tienen el potencial de aportar información adicional al ecosistema serverless, que evoluciona rápidamente.
- **Proyectos de prueba.** Se filtraron los proyectos que contienen cualquiera de las siguientes palabras clave en el nombre, la descripción o los temas: ("example", "tutorial", "demo", "sample", "starter", "playground", "hello-world", "test", "template", "learn", "workshop", "exercise", "skeleton", "boilerplate", "mock", "poc", "guide"). Esta lista se compiló a partir del código de Wonderless y la metodología de AWSomePy, además de 5 palabras clave añadidas tras un análisis exploratorio inicial (son las 5 últimas de la lista).

El uso de estos filtros adicionales resultó en un total de 816 repositorios.

2.5 Selección de proyectos con AWS como plataforma de destino

Solo se conservaron los proyectos destinados a AWS, ya que es la única plataforma compatible actualmente con Serverless Framework (para la versión 4, el *framework* dejó de admitir proveedores ajenos a AWS). Esto determinó el conteo final de 668 repositorios.

2.6 Clonación

Finalmente, se clonó la rama principal de cada repositorio para conservar las instantáneas asociadas al conjunto de datos. El tamaño total de los repositorios es de 51 GB y se ha publicado comprimido (6 GB) en Zenodo.

3

Caracterización del dataset

Se presenta un análisis de *OpenLambdaVerse* respecto al año de creación, el uso de *plugins*, la complejidad del código fuente, los entornos de ejecución (y lenguajes de programación), los temas, el tamaño de los repositorios, la popularidad, los *event triggers* y los metadatos relacionados con la seguridad en GitHub.

3.1 Año de creación

Dada la tendencia actual hacia el uso de IaC que habilite flujos de trabajo más programáticos, el uso de herramientas como Serverless Framework (basada en YAML) podría llegar a experimentar un declive gradual en su uso [20], es importante asegurar que *OpenLambdaVerse* no estuviera sesgado hacia repositorios más antiguos. La Figura 3.1 muestra el número de repositorios por año de creación. Aún no se observa una disminución en la popularidad del *framework*, ya que la caída en los repositorios observada en 2025 corresponde a un año parcial.

3.2 *Plugins*

Serverless Framework proporciona *plugins* para ampliar sus funcionalidades principales. La Tabla 3.1 describe los 11 *plugins* principales del conjunto de datos y el número de repositorios que utilizan cada uno. El *plugin* líder es `serverless-offline`, que emula AWS Lambda y API Gateway en un equipo local. Esto es útil para crear prototipos y, al mismo tiempo, evitar los costos de uso de la nube. El siguiente en popularidad es `serverless-webpack`, que se utiliza para agrupar funciones Lambda mediante Webpack; este complemento simula los puntos finales de la API local. De forma similar, para las pruebas locales, se usa `serverless-dynamodb-local`. `serverless-dotenv-plugin` gestiona los secretos en archivos .env, mientras que `serverless-plugin-typescript` y `serverless-python-requirements` son necesarios al usar Typescript y Python, respectivamente. El *plugin* `serverless-iam-roles-per-function` es para el control de acceso en AWS. Sin embargo, dado que está presente en solo 25 de 668 proyectos (3.59%), parece que las mejores prácticas para aplicar la gobernanza no se utilizan de forma

Figura 3.1: Años de creación de los proyectos en *OpenLambdaVerse* (fecha de corte: 24 de abril del 2025).

consistente en las plantillas de Serverless Framework (esto también se observó en AWSomePy [6]).

3.3 Complejidad de código fuente (medida a través de líneas de código, o LOC)

Se utiliza CLOC [21] para analizar las líneas de código (LOC) de los repositorios y así determinar cuáles son los lenguajes de programación más utilizados en el conjunto de datos (mostrado en la Tabla 3.2). Para ser coherentes con el análisis de los bytes de código realizado mediante la API de GitHub (descrito en 3.4), el análisis de complejidad de código tanto en esta sección como en 3.4 utiliza la lista de todos los lenguajes conocidos por GitHub [22], seleccionando aquellos con el tipo "programming".

La Figura 3.2 muestra la función de distribución acumulada (CDF) de las líneas de código en cada repositorio, con una media de LOC de 20,022. En cuanto a los lenguajes de programación en los repositorios (no solo los *handlers* de funciones), el principal es PHP (46.1%), seguido de JavaScript (38.1%), TypeScript (12.8%) y Python (1.4%), donde los porcentajes se calculan en base a las LOC de ese lenguaje respecto a las LOC totales en *OpenLambdaVerse*. Rust y Go son más populares que algunos lenguajes más tradicionales como C#, Java y Ruby, lo que pone de relieve su reciente aumento en popularidad y

Cuadro 3.1: Plugins de Serverless Framework más utilizados en *OpenLambaVerse*.

Plugin	Ocurrencias
serverless-offline	270
serverless-webpack	91
serverless-dotenv-plugin	87
serverless-plugin-typescript	80
serverless-python-requirements	71
serverless-prune-plugin	49
serverless-domain-manager	42
serverless-esbuild	31
serverless-bundle	25
serverless-iam-roles-per-function	25
serverless-dynamodb-local	24

adopción.

3.4 Bytes de código

La API REST de GitHub cuenta con un *endpoint* para listar los lenguajes de un repositorio y los bytes de código escritos en ellos [23]. Los valores agregados por lenguaje de programación se muestran en la Tabla 3.3.

PHP no es compatible de forma nativa con AWS Lambda, por lo que su alta prevalencia (52.4%) probablemente se deba a su uso para implementar páginas dinámicas; sin embargo, un pequeño porcentaje podría provenir de controladores, ya que el entorno de ejecución exclusivo del sistema operativo (proporcionado) puede utilizarse para ejecutar lenguajes no compatibles con AWS Lambda.

JavaScript (35.7%) y TypeScript (9.3%) son los lenguajes más populares compatibles con AWS Lambda en el conjunto de datos. La alta presencia de lenguajes no Lambda (bytes y LOC) evidencia las limitaciones del análisis automatizado de los lenguajes de programación presentes en los repositorios. Por lo tanto, para obtener una mejor idea de la prevalencia de los lenguajes en las funciones *serverless*, es recomendable analizar los entornos de ejecución utilizados (siguiente sección).

3.5 Entornos de ejecución

La Tabla 3.4 enumera los entornos de ejecución de funciones más populares. Node.js (JavaScript y TypeScript) es el entorno de ejecución líder, utilizado en el 73.58% de las funciones, seguido de Python (16.23%). Los entornos de ejecución solo para sistemas operativos (siempre que se indique) pueden utilizarse para lenguajes de programación no compatibles directamente con Lambda [24], como Go y Rust. El entorno de ejecución Go no compatible aparece en la tabla (2.07%) porque Lambda admitió

Cuadro 3.2: Líneas de código fuente por lenguaje de programación.

Lenguaje	Archivos	LOC	Porcentaje (%)
PHP	25,454	6,163,832	46.15%
JavaScript	16,522	5,081,974	38.05%
TypeScript	22,880	1,713,025	12.83%
Python	2,053	179,825	1.35%
Rust	49	74,902	0.56%
Go	698	57,893	0.43%
C#	269	19,805	0.15%
Cython	43	16,243	0.12%
Java	231	15,788	0.12%
Ruby	27	915	0.01%
PowerShell	2	160	<0.01%
Other	808	30,913	0.23%
Total	**69,036**	**13,355,275**	**100.00%**

el entorno de ejecución Go 1.x hasta enero de 2024 [25].

3.6 Tópicos

Los tópicos son etiquetas definidas por el usuario que se aplican a los repositorios de GitHub. La Tabla 3.5 muestra los 11 temas principales, incluyendo algunos que no sorprenden, como *serverless*, *aws-lambda*, *aws*, *serverless-framework* y *lambda*. Otros temas populares incluyen entornos de ejecución o lenguajes de programación (*node.js* y *typescript*) y un servicio específico de AWS (*dynamodb*). *hacktoberfest* es una competición anual organizada por GitHub; los proyectos que quedan después de filtrar son los que siguen activos, lo que destaca el impacto de este evento. *cvs-project* es el proyecto del Servicios de Vehículos Comerciales (CVS, por sus siglas en inglés: Commercial Vehicle Services) de la Agencia de Normas para Conductores y Vehículos (DVSA, por sus siglas en inglés: Driver and Vehicle Standards Agency) del Reino Unido.

3.7 Tamaño de repositorios

La Figura 3.3 muestra la CDF de los tamaños de repositorio, que varían entre 100 KB y 462 MB, con una mediana de 1.03 MB y una desviación estándar de 31 MB.

3.8 Contribuciones en los repositorios

El número de colaboradores, bifurcaciones (*forks*) e incidencias sirve como indicadores de la popularidad y adopción de un proyecto. Los repositorios de *OpenLambdaVerse* tienen un promedio de 5 colaboradores por proyecto, y el 34.58% de estos repositorios están gestionados por un solo colaborador. El conjunto de datos también revela un promedio de 7 incidencias abiertas por proyecto, lo que se traduce en un notable apoyo de la comunidad. Aunque el 43.71% de los proyectos no reportan nin-

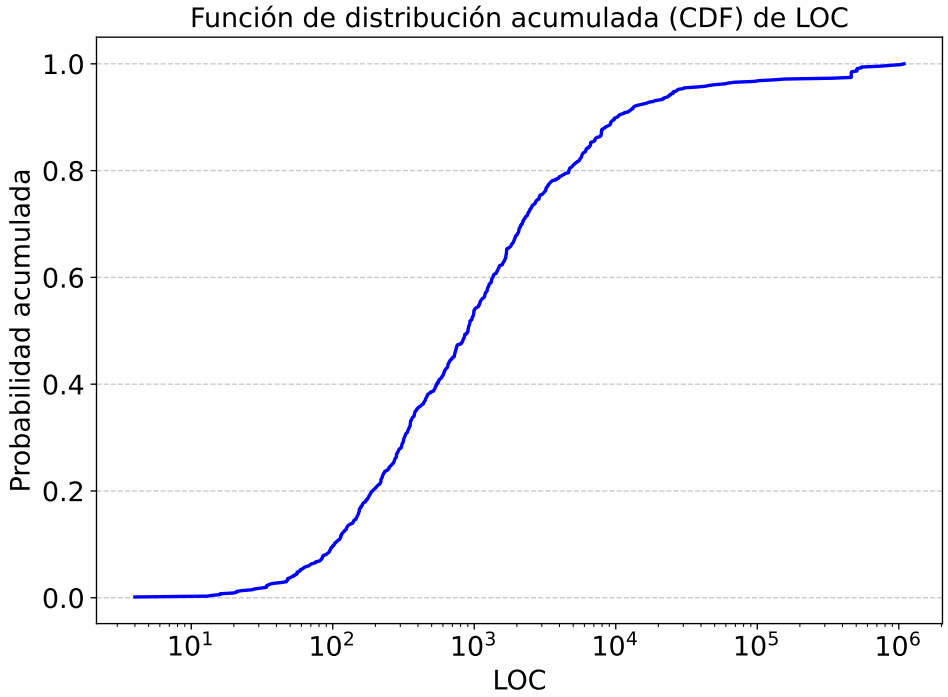

Figura 3.2: Distribución acumulada (escala logarítmica) de las LOC para todo el código en los repositorios (no sólo los *handlers* de funciones).

guna incidencia abierta, los metadatos recopilados por sí solos no permiten concluir si los proyectos inicialmente tenían alguna incidencia abierta. Además, una proporción significativa (38.62%) de estos repositorios se han bifurcado al menos una vez, otro indicador de la adopción general.

3.9 *Event triggers*

El código *serverless* se ejecuta en respuesta a eventos como solicitudes HTTP, tareas programadas y notificaciones. La Tabla 3.6 enumera estos disparadores y su prevalencia en el conjunto de datos; *http* y *httpApi* son los más populares, representando el 83.78% de los disparadores del conjunto de datos, y el 73.20% de los repositorios utilizan uno u otro al menos una vez.

Notar que tanto los *triggers http* como *httpApi* son disparadores HTTP/REST que provienen de API Gateway; el nombre se debe a un cambio de versión: v2 (httpApi) o v1 (http) [26].

Los eventos programados constituyen el 8.03% de los disparadores y aparecen en el 14.37% de los repositorios. Los *triggers* que aparecen en el 2-4% de los repositorios son: *sqs*, *s3* y *sns*.

La Figura 3.4 muestra la CDF del número de *triggers* por aplicación; en la mayoría de los proyectos (88.47%), el número de *triggers* está entre 1 y 4. A medida que el número de *triggers* supera este valor, se podría corroborar si esto representa una mayor complejidad del código en las implementaciones de funciones. La Figura 3.5 muestra la relación entre el número de *event triggers* y la complejidad del código (medida en LOC). Los resultados visuales y el coeficiente de correlación de 0.05 sugieren una

Cuadro 3.3: Lenguajes de programación más relevantes por tamano de código fuente (en bytes).

Lenguaje	Bytes totales	Aparición (%)
PHP	362,887,947	52.36%
JavaScript	247,084,664	35.65%
TypeScript	64,327,098	9.28%
Python	7,814,919	1.13%
Rust	2,773,185	0.40%
Go	1,962,996	0.28%
Cython	904,887	0.13%
ANTLR	857,375	0.12%
C#	830,584	0.12%
Java	772,276	0.11%
C	737,340	0.11%
Other	2,163,226	0.31%
Total	**693,116,497**	**100.00%**

Cuadro 3.4: Entornos de ejecución con mayor presencia en *OpenLambdaVerse*.

Entorno de ejecución	Conteo	Porcentaje
nodejs	426	73.58%
python	94	16.23%
provided (OS-only)	20	3.45%
java	16	2.76%
go	12	2.07%
ruby	7	1.21%
dotnet	4	0.69%

correlación insignificante entre las variables.

3.10 Metadatos de GitHub relacionados a la seguridad del repositorio

La comprobación de informes de vulnerabilidades privadas en un repositorio público es un metadato clave adicional relacionado con la seguridad del repositorio de GitHub [27]. Esta función es crucial para la comprobación y actualización constante de vulnerabilidades. Su implementación en tan solo 98 repositorios (14.67%) refuerza la preocupación por la falta de una correcta aplicación de las mejores prácticas de seguridad en aplicaciones *serverless*.

Cuadro 3.5: Tópicos con mayor presencia en el *dataset*.

Tópico	Conteo
`serverless`	92
`aws-lambda`	59
`aws`	42
`serverless-framework`	36
`lambda`	36
`nodejs`	32
`typescript`	30
`cvs-project`	17
`hacktoberfest`	16
`dynamodb`	15

Cuadro 3.6: *Event triggers* utilizados para llamar a las funciones *serverless*.

Event trigger	Conteo	Triggers (%)	Repos (%)
http	1547	62.43%	51.05%
httpApi	529	21.35%	23.20%
schedule	199	8.03%	14.37%
websocket	48	1.94%	1.80%
sqs	30	1.21%	3.14%
sns	27	1.09%	2.40%
s3	24	0.97%	2.69%
eventBridge	17	0.69%	0.90%
stream	11	0.44%	1.65%
cloudwatchEvent	9	0.36%	1.05%
cognitoUserPool	6	0.24%	0.45%
cloudwatchLog	4	0.16%	0.45%
alexaSkill	1	0.04%	0.15%
alb	1	0.04%	0.15%

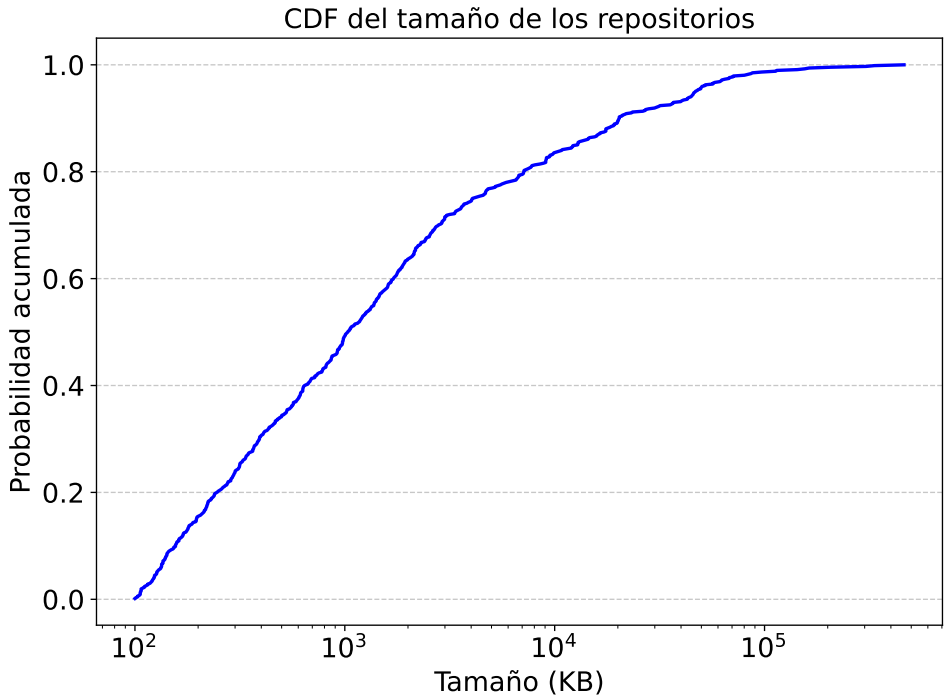

Figura 3.3: Distribución acumulada (escala logarítmica) del tamaño de los repositorios.

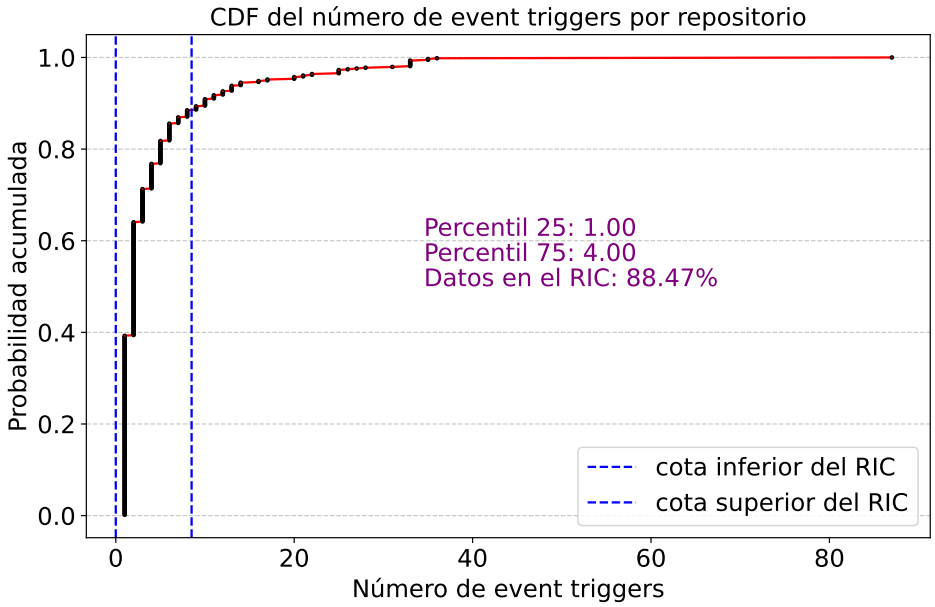

Figura 3.4: Distribución acumulada del número de *event triggers* por repositorio.

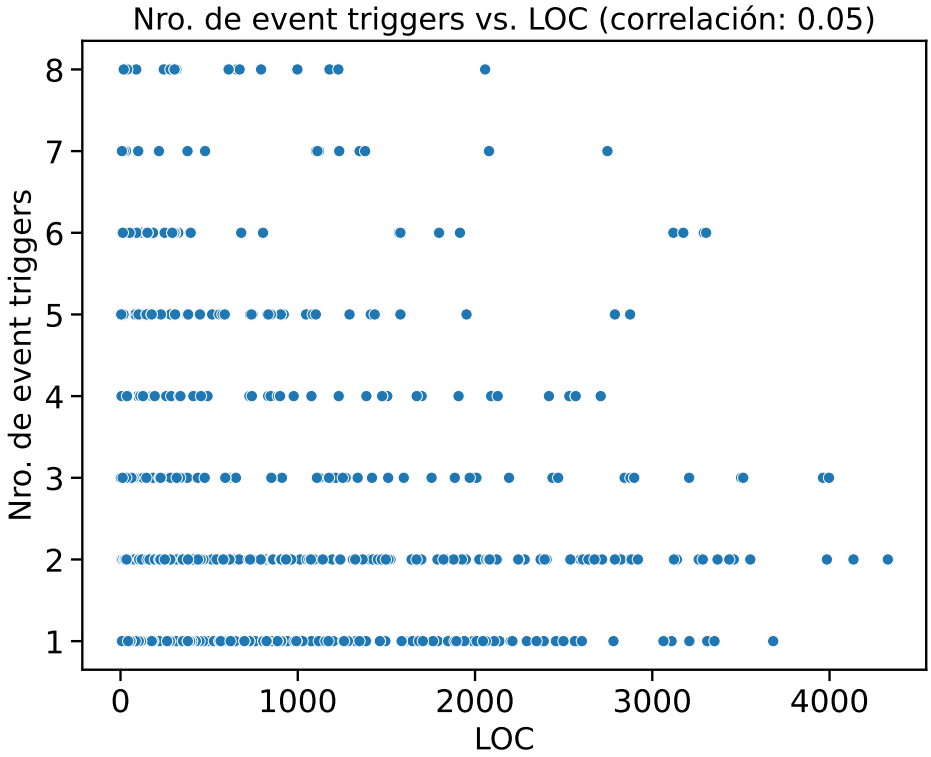

Figura 3.5: Número de event *triggers* vs. LOC

4

Comparación con trabajo previo

A continuación, se compara la caracterización realizada con trabajos previos en el ámbito de las aplicaciones *serverless*. La Tabla 4.1 enumera los estudios analizados. Además de estos estudios, también se remite al informe "State of Serverless" de Datadog [20], que obtiene datos de más de 20,000 empresas de su base de clientes y se actualizó anualmente entre 2020 y 2023. No se incluye en la tabla porque sus datos fuente no están disponibles públicamente. No incluimos estudios basados en trazas que estudian/compilan llamadas *serverless* sin analizar las aplicaciones en sí (por ejemplo, el estudio "Serverless in the Wild" [28]).

Con la excepción del trabajo de Eismann et al. [4], los demás estudios de la Tabla 4.1 se centran en Serverless Framework debido a su amplia adopción. Este *framework* fue uno de los primeros productos en este ámbito y se utiliza ampliamente para la implementación de aplicaciones *serverless* en AWS [20, 29]. A diferencia de los otros estudios de la tabla, Eismann et al. realizaron un análisis manual de cada una de las 89 aplicaciones estudiadas. Este enfoque manual les permitió considerar cualquier tipo de aplicación serverless, a costa de ser el estudio que abarcó el menor número de aplicaciones. Wonderless también consideró proveedores más allá de AWS; sin embargo, para *OpenLambdaVerse* no consideramos otras plataformas, ya que Serverless Framework dejó de ser compatible con proveedores

Cuadro 4.1: Datasets de aplicaciones *serverless* y caracterizaciones realizadas.

Estudio y año de recolección	# de aplicaciones	Descripción del dataset	URL del dataset
Wonderless [3] 2020	1,877	URLs de repositorios y repositorios clonados (GitHub, Serverless Framework: AWS, Azure, Google, OpenWhisk, Cloudflare, Kubeless).	https://zenodo.org/records/4451387
Eismann et al. [4] 2021	89	Análisis tabulado de aplicaciones *serverless* obtenidas de GitHub, literatura académica, y literatura industrial; AWS, Azure, IBM, Google & nube privada.	https://zenodo.org/records/5185055
AWSomePy [6] 2023	145	Resumen del *dataset* (metadata) y 145 repositorios clonados (GitHub, Python, Serverless Framework: AWS).	https://zenodo.org/records/7838077
OpenLambdaVerse [13]2025	668	Metadata tabulada y repositorios clonados (GitHub, Serverless Framework: AWS).	https://zenodo.org/records/16533581

ajenos a AWS con el lanzamiento de la versión 4 en mayo de 2024 [30].[1].

Dado que estos estudios se centran en diferentes *frameworks* y proveedores de la nube, no se pueden extraer conclusiones sólidas al comparar los resultados de las diferentes caracterizaciones de las aplicaciones *serverless*. No obstante, se detallan las diferencias entre nuestros resultados y estudios anteriores, sin hacer afirmaciones contundentes sobre el significado de las diferencias.

4.1 Entornos de ejecución

En comparación con las estadísticas de Wonderless, la popularidad de los entornos de ejecución de Node.js (73.6% frente a 72.2%) y Python (16.2% frente a 19%) se ha mantenido relativamente estable, con un ligero aumento de popularidad de Node.js y una disminución de la de Python. Estos dos entornos de ejecución también se reportan como los más comunes en el estudio de Eismann (ambos con un 42%) y en el estudio de Datadog (aproximadamente un 43% de Node.js y un 28% de Python), aunque los porcentajes de prevalencia difieren debido a las diversas metodologías de inclusión de aplicaciones. En resumen, todos los estudios concluyeron que los entornos de ejecución más populares para funciones serverless son Node.js y Python, seguidos de Java en un distante tercer lugar.

En OpenLambdaVerse y Wonderless, los demás entornos de ejecución tienen una popularidad inferior al 3.5% y se han mantenido estables en los últimos 5 años. Sin embargo, tanto Eismann como Datadog informan que los entornos de ejecución de Java tienen una mayor prevalencia (12% y 10%); este lenguaje también ocupa el tercer lugar en popularidad en OpenLambdaVerse y Wonderless, pero con una popularidad menor (2.8%). La diferencia en las cifras de Datadog es particularmente importante, dado que también las informan solo para AWS, por lo que el soporte de diferentes proveedores no puede explicarlas. Creemos que esta diferencia (mayor popularidad de Java en el estudio de Datadog) puede deberse a la inclusión de repositorios privados o al hecho de que también incluyeron repositorios con aplicaciones *serverless* gestionadas por Terraform, que, según informan, es la herramienta de implementación de AWS Lambda preferida entre las organizaciones más grandes (lo que, a su vez, podría favorecer a implementaciones realizadas en Java).

Al comparar OpenLambdaVerse con Wonderless, observamos una disminución de 15% en el uso de los entornos de ejecución de Python y Go, probablemente como resultado de la descontinuación de este entorno de ejecución por parte de AWS en enero de 2024 [25]. El único entorno de ejecución con un aumento significativo en su adopción es "provided" (8%), que podría haber experimentado un aumento en su adopción para la compatibilidad con Go, ya que Lambda ya no admite este lenguaje. Datadog también descubrió que los entornos de ejecución personalizados son el tipo de funciones de mayor crecimiento y postula que esto podría deberse a un mayor interés en los contenedores serverless.

4.2 *Plugins*

AWSomePy es el único otro estudio que incluye la caracterización del uso de *plugins*. En *OpenLambaVerse*, los tres plugins principales son `serverless-offline` (40% de prevalencia), `serverless-webpack` (14% de prevalencia) y `serverless-dotenvplugin` (13% de prevalencia). Estos resultados difieren de los resultados de AWSomePy: `serverless-python-requirements` (65%), `serverless-pseudo-parameters`

[1] https://github.com/serverless/serverless/releases?q=4.0&expanded=true

(17%) y `serverless-domainmanager` (10%), presumiblemente porque AWSomePy solo tiene aplicaciones Python. En *OpenLambdaVerse*, `serverless-pythonrequirements` aparece en quinto lugar, con una prevalencia del 10%.

4.3 Complejidad (LOC)

Wonderless y AWSomePy también informan sobre las líneas de código (LOC) por lenguaje y por repositorio, respectivamente. Sin embargo, la Tabla II no es comparable con la Tabla 1 del artículo de Wonderless [3] porque su análisis se realiza únicamente para los controladores; por lo tanto, presentamos una comparación entre su Tabla 1 y la Tabla IV en el apartado "Entornos de ejecución" de esta sección. Con respecto a las líneas de código, *OpenLambdaVerse* tiene un promedio de LOC por repositorio de 20,022, con un 53.82% de los repositorios con 1000 LOC o menos, y un 9.75% con 100 LOC o menos.

En AWSomePy, los repositorios son en promedio más pequeños (4468 LOC), pero tienen un LOC similar para los percentiles 10 y 55 (los únicos percentiles reportados específicamente en [6]): el 55% de los repositorios tiene menos de 1000 LOC y el 10% menos de 100.

4.4 *Event triggers*

Solo el estudio de Eismann (E) analiza qué tipos de *triggers* se utilizan para activar funciones *serverless*. En su estudio y en *OpenLambdaVerse* (O), los tres *triggers* más populares son HTTP, eventos en la nube (suma de *triggers* provenientes de otros eventos de servicio como SQS, SNS y S3) y programados. Sin embargo, las cifras específicas difieren: HTTP (O: 86%, E: 48%), programados (O: 16%, E: 13%), eventos en la nube (O: 12%, E: 41%). Estas diferencias podrían no representar cambios en las prácticas de desarrollo, sino más bien el resultado de diferentes metodologías de selección y análisis de las aplicaciones utilizadas por Eismann et al. (es decir, sin *scraping* ni análisis automatizados).

5

Discusión

Un importante hallazgo en este estudio es que las funciones *serverless* se encuentran frecuentemente presentes en proyectos grandes que involucran otros componentes no *serverless* (por ejemplo, páginas web dinámicas en PHP), con funciones escritas predominantemente en Node.js (JavaScript y Typescript) y Python, y activadas principalmente por eventos http y programados. Esto se interpreta como un indicador de que las funciones *serverless* se utilizan con frecuencia en proyectos más grandes, no completamente *serverless*, debido a su simplicidad y facilidad de implementación.

En cuanto a los tipos de *triggers* utilizados para llamar a las Lambdas, se observa que faltan *triggers* específicos de IoT, una observación sorprendente dado que Lambda se utiliza en un número significativo de aplicaciones IoT/Edge: Un estudio reciente descubrió que Lambda está presente en el 60% de las aplicaciones de arquitecturas Edge en un conjunto de datos reciente basado en AWS [31]. Sin embargo, las aplicaciones de ese conjunto de datos pueden activar Lambda a través de otros mecanismos como http, stream (Kinesis), websockets e indirectamente a través de otros servicios en la nube. Para un ejemplo de esto, véase la Fig. 1 en [31] (reproducida aquí para mayor claridad, en la Figura 5.1). Además, es posible que las aplicaciones de IoT en GitHub sean, en un número significativo de casos, de código cerrado o sin licencia y, por lo tanto, excluidas de los pasos de filtrado, o que prefieran otros *frameworks* de IaC como Terraform [20] o incluso otros proveedores como Cloudflare.

La baja tasa de implementación de la función de informes de vulnerabilidades privadas de GitHub genera preocupación por la divulgación de vulnerabilidades críticas en estas aplicaciones. Los propietarios de los repositorios deberían implementar herramientas que les permitan abordar los problemas de seguridad de forma privada antes de que se hagan públicos. Además, en este documento no se estudia la fiabilidad de los *plugins* y bibliotecas utilizados en los proyectos, una preocupación creciente en aplicaciones que dependen de paquetes externos [32]; estos plugins de seguridad siempre deben evaluarse exhaustivamente para detectar posibles vulnerabilidades antes de su adopción.

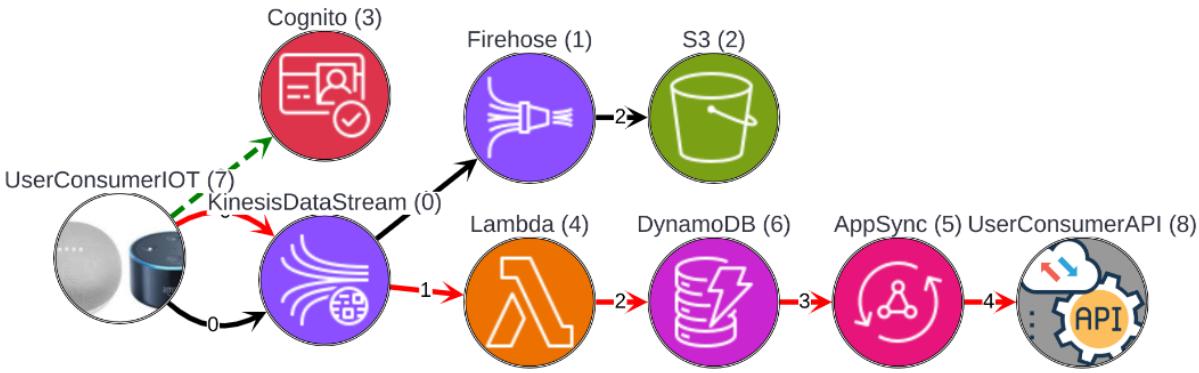

Figura 5.1: Arquitectura IoT de ejemplo del dataset de Cloudscape [31]: TechConnect (deportes impulsados por tecnología con aprendizaje automático de IoT).

6

Amenazas a la validez

Si bien el proceso de recopilación y análisis de datos se diseñó para ser sistemático (y reproducible), el presente estudio posee limitaciones que podrían afectar la generalización e interpretación de los hallazgos. Estas amenazas se categorizan en validez interna y validez externa.

6.1 Validez externa

6.1.1 Alcance y representatividad del dataset

El *dataset* está compuesto por repositorios públicos. Sin embargo, una parte sustancial del desarrollo de software, en particular para aplicaciones propietarias y empresariales, se realiza en repositorios privados. La inaccesibilidad de estos datos limita la profundidad y amplitud del análisis, lo que podría introducir un sesgo hacia las prácticas predominantes en proyectos de código abierto. Por lo tanto, las tendencias y características identificadas en este estudio podrían no representar el panorama completo del uso de funciones *serverless*, especialmente en entornos de producción dentro de organizaciones que utilizan intensivamente repositorios privados.

6.1.2 *Stack* elegido (Serverless Framework + AWS Lambda)

Una amenaza significativa para la generalización de los hallazgos proviene del enfoque particular en repositorios que utilizan Serverless Framework y AWS Lambda. El ecosistema *serverless* es diverso e incluye múltiples *frameworks* (incluyendo opciones de código abierto como OpenWhisk) y diversos proveedores de nube (p. ej., Azure Functions, Google Cloud Functions, Cloudflare Workers). Sin embargo, la elección del proveedor de nube se basa en que AWS continúa siendo el proveedor con la mayor adopción en computación *serverless* [20]. El uso de los archivos de configuración de Serverless Framework para identificar proyectos *serverless* y la selección de AWS como el proveedor de nube implica que los resultados podrían no ser directamente transferibles a proyectos creados con otros *frameworks* o implementados en diferentes plataformas de nube. En el caso de AWS, este *framework* sigue siendo la

herramienta de IaC más utilizada para gestionar funciones de AWS Lambda [20], pero Terraform es más popular para arquitecturas altamente complejas [20], por lo que no considerarlo sesga los resultados. Además, los nuevos proyectos prefieren cada vez más las opciones nativas de AWS, como CDK, SAM y Chalice [20], y no incluirlas en el presente estudio también limita la generalización de los resultados. Los diferentes *frameworks* ofrecen distintas características, convenciones y niveles de abstracción, lo que podría influir en la estructura, la seguridad y la implementación de las aplicaciones *serverless*. De igual manera, cada proveedor de nube cuenta con servicios, prácticas recomendadas y modelos de seguridad únicos. Si bien AWS Lambda es una plataforma *serverless* destacada, una comparación más amplia que incluya otros *frameworks* o plataformas de código abierto mejoraría la generalización de los hallazgos.

6.2 Validez interna

6.2.1 Criterio y limitaciones del filtrado de repositorios

La precisión de los hallazgos reportados en esta investigación depende de la efectividad de los filtros utilizados para identificar repositorios que emplean funciones *serverless* (y que no son proyectos de prueba, demostraciones, etc.). Para minimizar este sesgo, los valores de corte establecidos en los criterios de selección se fundamentan en las mismas prácticas utilizadas por la comunidad de MSR (y haciendo referencia a las respectivas fuentes cuando correspondía), pero este proceso automatizado podría excluir inadvertidamente repositorios relevantes o incluir proyectos irrelevantes que logren superar estos umbrales.

6.2.2 Interpretación de prácticas de seguridad a partir del análisis de los metadatos

Si bien el análisis destacó la preocupación por el bajo uso de complementos de seguridad en los repositorios minados, se reconoce un problema con este enfoque: las funciones de seguridad en las aplicaciones en la nube se pueden gestionar directamente a través de la interfaz web del proveedor de la nube. Por lo tanto, la ausencia de configuraciones de seguridad explícitas en el repositorio de código no implica necesariamente una falta de buenas prácticas de seguridad en la aplicación implementada (aunque se puede argumentar que no incluir dichas configuraciones en los repositorios es propenso a errores manuales y constituye un problema de seguridad en sí mismo).

7

Conclusiones

En este trabajo se ha presentado *OpenLambdaVerse*, un conjunto de datos de 668 repositorios, obtenidos mediante una metodología de *scraping* refinada que integra las mejores prácticas de la comunidad de MSR y extiende la propuesta original de *Wonderless*.

La caracterización del conjunto de datos realizada en esta investigación supera el alcance de la realizada en Wonderless y sus estudios derivados, ofreciendo una visión actualizada de las aplicaciones *serverless* en el mundo real. El análisis de los metadatos obtenidos de GitHub y de los archivos de configuración de *Serverless Framework* proporciona una perspectiva más completa sobre la arquitectura de estas aplicaciones, considerando aspectos técnicos previamente inexplorados.

Tanto el *dataset* como la caracterización constituyen un recurso valioso que puede ser utilizado por profesionales y académicos para comprender mejor la evolución de las cargas de trabajo *serverless*. La publicación de estos artefactos, permite que otros miembros de la comunidad puedan aprovechar y extender este trabajo para obtener una comprensión más profunda del panorama cambiante de las aplicaciones *serverless*.

Referencias

[1] S. Kounev, N. Herbst, C. L. Abad, A. Iosup, I. Foster, P. Shenoy, O. Rana, and A. A. Chien, "Serverless computing: What it is, and what it is not?" *Commun. ACM*, vol. 66, no. 9, p. 80–92, Aug. 2023.

[2] "Serverless Framework," https://www.serverless.com/, accedido el 8 de mayo del 2025.

[3] N. Eskandani and G. Salvaneschi, "The Wonderless Dataset for Serverless Computing," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 565–569.

[4] S. Eismann, J. Scheuner, E. v. Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "The state of serverless applications: Collection, characterization, and community consensus," *IEEE Transactions on Software Engineering*, vol. 48, no. 10, 2022.

[5] S. Bhatnagar, Z. Li, Z. Song, and E. Tilevich, "Os^3: The art and the practice of searching for open-source serverless functions," in *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2023.

[6] G. Raffa, J. Alis, D. O'Keeffe, and S. Dash, "AWSomePy: A Dataset and Characterization of Serverless Applications," in *Workshop on SErverless Systems, Applications and MEthodologies (SESAME)*, 2023.

[7] "Wonderless," https://zenodo.org/records/4451387, accessed on May 5, 2025.

[8] "Wonderless implementation," https://github.com/prg-grp/wonderless/tree/master/impl, accessed on May 5, 2025.

[9] "serverless□dataset," https://github.com/edgeumd/serverless□dataset, accessed on May 5, 2025.

[10] "AWSomePy dataset," https://doi.org/10.5281/zenodo.7838076, accessed on May 5, 2025.

[11] GitHub, "Repository search results • GitHub," En línea. Disponible en: https://github.com/search?q=is%3Apublic&type=Repositories, 2026, accedido el 27 de enero del 2026.

[12] Dan, "GitCharts - GitHub Statistics," En línea. Disponible en: https://gitcharts.com/, 2026, accedido el 27 de enero del 2026.

[13] A. C. Chavez-Moreno and C. L. Abad, "Openlambdaverse: A dataset and analysis of open-source serverless applications," 2025. [Online]. Available: https://arxiv.org/abs/2508.01492

[14] Y. Sens, H. Knopp, S. Peldszus, and T. Berger, " A Large-Scale Study of Model Integration in ML-Enabled Software Systems ," in *IEEE/ACM International Conference on Software Engineering (ICSE)*, May 2025.

[15] H. He, B. Vasilescu, and C. Kästner, "Pinning is futile: You need more than local dependency versioning to defend against supply chain attacks," in *ACM Foundations of Software Engineering (FSE)*, 2025.

[16] S. Baltes, J. Knack, D. Anastasiou, R. Tymann, and S. Diehl, "(no) influence of continuous integration on the commit activity in github projects," in *ACM SIGSOFT International Workshop on Software Analytics*, 2018.

[17] J. Coelho, M. Valente, L. Silva, and E. Shihab, "Identifying unmaintained projects in github," in *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018.

[18] W. Lu, E. Kasaadah, S. Karim, M. Germonprez, and S. Goggins, "Open source software lifecycle classification: Developing wrangling techniques for complex sociotechnical systems," *arXiv preprint arXiv:2504.16670*, 2025.

[19] S. Flint, J. Chauhan, and R. Dyer, "Pitfalls and guidelines for using time-based git data," *Empirical Software Eng.*, vol. 27, no. 7, 2022.

[20] Datadog, "The state of serverless," 2023, online, last accessed on Jul. 27, 2025. [Online]. Available: https://www.datadoghq.com/state-of-serverless/

[21] A. Danial, "cloc (GitHub Repository)," https://github.com/AlDanial/cloc, accessed on August 28, 2023.

[22] GitHub, "Archivo `languages.yml` del repositorio *Linguist*," En línea. Disponible en: https://github.com/github-linguist/linguist/blob/main/lib/linguist/languages.yml, 2026, accedido el 27 de enero del 2026.

[23] "GitHub: List repository languages," https://docs.github.com/en/rest/repos/repos?apiVersion=2022-11-28#list-repository-languages, accessed on May 5, 2025.

[24] AWS, "When to use Lambda's OS-only runtimes," 2025, online, last accessed on Jul. 28, 2025. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/runtimes-provided.html

[25] ——, "Lambda runtimes," 2025, online, last accessed on Jul. 30, 2025. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html

[26] I. Serverless, "Http api (api gateway v2)," 2025, online, last accessed on Jul. 28, 2025. [Online]. Available: https://www.serverless.com/framework/docs/providers/aws/events/http-api

[27] "GitHub: Check if private vulnerability reporting is enabled for a repository," https://docs.github.com/en/rest/repos/repos?apiVersion=2022-11-28#check-if-private-vulnerability-reporting-is-enabled-for-a-repository, accessed on May 5, 2025.

[28] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *USENIX Annual Technical Conference (USENIX ATC)*, 2020.

[29] K. Kritikos and P. Skrzypek, "A review of serverless frameworks," in *IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018.

[30] I. Serverless, "Upgrading to serverless framework v4: Deprecation of non-aws providers," 2025, online, last accessed on Jul. 27, 2025. [Online]. Available: https://www.serverless.com/framework/docs/guides/upgrading-v4#deprecation-of-non-aws-providers

[31] S. Santillan and C. Abad, "An analysis of hpc and edge architectures in the cloud," in *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E, to appear), 2nd Workshop on Accelerated HPC in the Cloud-Edge Continuum*, 2025.

[32] M. Alfadel, D. E. Costa, E. Shihab, and B. Adams, "On the discoverability of npm vulnerabilities in node.js projects," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 4, May 2023.

Archivos adjuntos

Anexo A

Código fuente del script desarrollado para la búsqueda de repositorios en GitHub

Este anexo presenta el script desarrollado en *Bash* para la búsqueda y extracción de repositorios en GitHub que utilizan *Serverless Framework*. El script implementa un algoritmo de ajuste dinámico de intervalos basado en el tamaño de los archivos para mitigar las restricciones de paginación de la API de GitHub.

```bash
#!/usr/bin/env bash

url="https://api.github.com" # GitHub REST API base URL

# Source the .env file and set the GitHub token
if [ -f ".env" ]; then
  source .env
fi

if [ -z "$GITHUB_AUTH_TOKEN" ]; then
  echo "Error: GITHUB_AUTH_TOKEN is not set in your .env file."
  exit 1
fi

token="$GITHUB_AUTH_TOKEN"

today=$(date +"%Y-%m-%d") # Current date for filenames

# Filename-based code search
config_files=(
    "serverless.yml" # Serverless Framework
    # Add other platform/framework specific files if needed
)

# We define the base directory for the code search
parent_dir="data/raw/code_search_$(date +"%Y%m%d_%H%M%S")" # The timestamp guarantees uniqueness per run

# We define subdirectories for the code search results, errors, and logs
results_dir="$parent_dir/results_by_filename"
errors_dir="$parent_dir/errors"
logs_dir="$parent_dir/logs"

# Log file path
log_file="$logs_dir/code_search_${today}.log"

# To comply with GitHub API rate limits, we will implement dynamic interval adjustment, starting with a
↪  moderate interval and then adjusting based on results

# File size bounds (in bytes)
lower_bound=0 # 0 bytes (adjust as needed)
upper_bound=384000 # 384KB (current limit for searchable files, adjust as needed)

# Interval settings
min_interval=20 # Start with a small interval for file sizes (adjust as needed)
max_interval=100000 # Largest allowed file size interval (adjust as needed)
current_interval=$min_interval # Start with the previously defined minimum
```

```
46
47  # Thresholds for adjusting interval (adjust as needed)
48  low_results_threshold=900
49  high_results_threshold=900 # Consider decreasing if > this OR API limit hit
50
51  # Function to check for required dependencies
52  dependency_test()
53  {
54    echo "Checking dependencies..."
55    for dep in curl jq sed sort wc tr mkdir; do # Added mkdir
56      if ! command -v "$dep" &> /dev/null; then
57        echo -e "\nError: I require the '$dep' command but it's not installed.\n"
58        exit 1
59      fi
60    done
61    echo "All dependencies found."
62  }
63
64  # Function to validate GitHub token
65  token_test()
66  {
67    if [ -z "$token" ]; then
68      echo "Error: You must set a Personal Access Token to the GITHUB_AUTH_TOKEN environment variable or in a
      ↪  .env file."
69      exit 1
70    fi
71    http_status=$(curl --silent --output /dev/null --write-out "%{http_code}" -H "Authorization: token
      ↪  $token" "$url/user")
72    if [ "$http_status" -ne 200 ]; then
73        echo "Error: GitHub token seems invalid or lacks permissions (HTTP status: $http_status). Please
          ↪  check it and try again."
74        exit 1
75    else
76        echo "GitHub token validated."
77        token_cmd="Authorization: token $token"
78    fi
79  }
80
81  # Progress indicators
82  working() {
83    printf "Fetching page..."
84  }
85
86  work_done() {
87    printf " done!\n"
88  }
89
90  # Function to append results to the output file
91  output_list() {
92      local current_filename=$1
93      local size_range=$2
94      local batch_array_name=$3
95      local target_file=$4
96
97      local items_ref="${batch_array_name}[@]"
98      local items=("${!items_ref}")
99      local count=${#items[@]}
100
```

```
    if [ "$count" -gt 0 ]; then
        echo "# Results for: $current_filename (size: $size_range)" >> "$target_file"
        printf '%s\n' "${items[@]}" >> "$target_file"
        echo "" >> "$target_file"
    fi
    echo $count # Return count
}

# Function to fetch repos for a specific filename
get_repos_for_file() {
  local current_filename=$1
  echo "Starting search for filename: $current_filename"

  # Define and Clear Specific Output File
  local base_filename="${current_filename}_results_${today}.txt"
  base_filename=$(echo "$base_filename" | sed 's/[^a-zA-Z0-9._-]/_/g') # Sanitize
  local specific_output_file="$results_dir/$base_filename"

  echo "Clearing/creating output file for this config: $specific_output_file"
  # Ensure directory exists before writing
  mkdir -p "$(dirname "$specific_output_file")"
  > "$specific_output_file"

  # Reset interval and start point for each file
  current_interval=$min_interval
  local i=$lower_bound

  while [[ $i -le $upper_bound ]]; do
    local hit_api_limit=0
    local batch_count=0
    local max_page=0

    local j=$((i + current_interval - 1))
    if [[ $j -gt $upper_bound ]]; then j=$upper_bound; fi
    if [[ $i -gt $j ]]; then
        echo "Warning: Calculated start $i exceeds end $j. Finishing search for $current_filename."
        break
    fi
    local size_range="$i..$j"
    echo "Searching size range: $size_range bytes (interval: $current_interval) for $current_filename"
    sleep 8 # API rate limit precaution

    local query="filename:$current_filename+size:$size_range"
    local api_endpoint="$url/search/code?q=$query&per_page=100"

    # Check Pagination
    local last_repo_page=$(curl --silent --head -H "$token_cmd" "$api_endpoint" | sed -nE
    ↪  's/^link:.*page=([0-9]+)>; rel="last".*/\1/p')
    if [[ $? -ne 0 ]]; then
        echo "Error checking pagination for $current_filename size $size_range. Skipping range."
        i=$((i + current_interval))
        continue
    fi

    # Preemptive Retry Optimization
    local potential_max_page=${last_repo_page:-1}
    local requires_retry_immediately=0
    if [[ "$potential_max_page" -ge 10 ]]; then requires_retry_immediately=1; fi
```

```
158
159     if [[ $requires_retry_immediately -eq 1 ]]; then
160         echo "Optimization: Detected >= 10 pages needed for interval $current_interval (range
            ↪  $size_range)."
161         local next_interval=$((current_interval / 2))
162         if [[ $next_interval -lt $min_interval ]]; then next_interval=$min_interval; fi
163
164         if [[ $next_interval -lt $current_interval ]]; then
165             echo "Reducing interval to $next_interval and retrying range starting at $i immediately."
166             current_interval=$next_interval
167             continue # Skip fetching for the current wide interval
168         else
169             echo "Warning: >= 10 pages required, but interval is already at minimum ($min_interval).
                ↪  Proceeding to fetch results for $size_range (may be incomplete)."
170             hit_api_limit=1
171             max_page=10
172         fi
173     else
174         max_page=$potential_max_page
175         if [[ "$max_page" -gt 10 ]]; then max_page=10; fi
176     fi
177
178     # Proceed with fetching pages
179     local repos_batch=() # Local array for the batch
180
181     if [[ -z "$last_repo_page" ]] && [[ $requires_retry_immediately -ne 1 ]]; then
182       # Single page results or no pagination
183       echo "Fetching single page or no results for $current_filename size $size_range"
184       sleep 8
185       local response=$(curl --silent -H "$token_cmd" "$api_endpoint")
186       if [[ $? -ne 0 ]]; then
187           echo "Error fetching results for $current_filename size $size_range. Skipping range."
188           i=$((i + current_interval))
189           continue
190       fi
191
192       if ! echo "$response" | jq -e . > /dev/null 2>&1; then
193           echo "ERROR: Invalid JSON received for $current_filename size $size_range PAGE 1. Skipping
              ↪  range."
194           local error_base_filename="error_response_${current_filename}_${size_range}_page1_$(date
              ↪  +%s).json"
195           local error_filepath="$errors_dir/$error_base_filename"
196           echo "Saving invalid JSON to: $error_filepath"
197           mkdir -p "$(dirname "$error_filepath")" # Ensure error dir exists
198           echo "$response" > "$error_filepath"
199           i=$((i + current_interval))
200           continue
201       fi
202
203       local paginated_repos_str=$(echo "$response" | jq --raw-output 'if .items then .items[].html_url else
          ↪  empty end // empty')
204
205       repos_batch=()
206       while IFS= read -r line; do if [[ -n "$line" ]]; then repos_batch+=("$line"); fi; done <<<
          ↪  "$paginated_repos_str"
207       if [[ ${#repos_batch[@]} -eq 0 ]]; then echo "No repositories found in this batch."; fi
208       hit_api_limit=0
209
```

```
    elif [[ $max_page -gt 0 ]]; then
      # Multiple pages (up to 10)
      if [[ "$max_page" -eq 10 ]]; then hit_api_limit=1; fi
      echo "Fetching $max_page pages for $current_filename size $size_range"
      for (( k=1; k<=$max_page; k++ )); do
        working
        sleep 8
        local response=$(curl --silent -H "$token_cmd" "$api_endpoint&page=$k")
        if [[ $? -ne 0 ]]; then echo "Error fetching page $k for $current_filename size $size_range.
        ↪  Skipping page."; continue; fi

        if ! echo "$response" | jq -e . > /dev/null 2>&1; then
            echo "ERROR: Invalid JSON on page $k for $current_filename size $size_range. Skipping page."
            local error_base_filename="error_response_${current_filename}_${size_range}_page${k}_$(date
            ↪  +%s).json"
            local error_filepath="$errors_dir/$error_base_filename"
            echo "Saving invalid JSON to: $error_filepath"
            mkdir -p "$(dirname "$error_filepath")" # Ensure error dir exists
            echo "$response" > "$error_filepath"
            continue
        fi

        local paginated_repos_str=$(echo "$response" | jq --raw-output 'if .items then .items[].html_url
        ↪  else empty end // empty')

        local page_batch=()
        while IFS= read -r line; do if [[ -n "$line" ]]; then page_batch+=("$line"); fi; done <<<
        ↪  "$paginated_repos_str"
        if [[ ${#page_batch[@]} -gt 0 ]]; then repos_batch+=("${page_batch[@]}"); else echo "No
        ↪  repositories found on page $k."; fi
      done
      work_done
    fi # End fetching logic

    # Output results for this batch
    batch_count=$(output_list "$current_filename" "$size_range" "repos_batch" "$specific_output_file")
    if [[ "$batch_count" -gt 0 ]]; then
        echo "Appended $batch_count repository URLs for $current_filename (size: $size_range) to
        ↪  $specific_output_file"
    fi

    # Adjusting interval based on results
    local next_interval=$current_interval
    if [[ $batch_count -lt $low_results_threshold ]] && [[ $hit_api_limit -ne 1 ]]; then
        next_interval=$((current_interval * 2))
        if [[ $next_interval -gt $max_interval ]]; then next_interval=$max_interval; fi
        if [[ $next_interval -ne $current_interval ]]; then
            echo "Adjusting interval: Low results ($batch_count). Increasing interval for next search to
            ↪  $next_interval"
        fi
    # Optional: Add logic here to decrease interval if hit_api_limit was 1 and interval > min_interval
    elif [[ $hit_api_limit -eq 1 ]] && [[ $current_interval -gt $min_interval ]]; then
         next_interval=$((current_interval / 2))
         if [[ $next_interval -lt $min_interval ]]; then next_interval=$min_interval; fi
         if [[ $next_interval -ne $current_interval ]]; then
             echo "Adjusting interval: Hit API limit. Decreasing interval for next search to
             ↪  $next_interval"
```

```
        fi
    fi

    # Calculate the next starting point
    local interval_used_this_iteration=$current_interval
    i=$((i + interval_used_this_iteration))
    current_interval=$next_interval

  done # End while loop

  echo "Finished search for filename: $current_filename"
  echo ""
} # End function get_repos_for_file


#### MAIN

# Initial checks output to console
dependency_test
token_test

# Create directories first - this echo will go to console
echo "Ensuring directories exist: $results_dir, $errors_dir, and $logs_dir"
mkdir -p "$results_dir" "$errors_dir" "$logs_dir"

# Setup log file - redirect all output from here on to the log file
exec > "$log_file" 2>&1
echo "Log started at $(date)"
echo "Full log is being written to: $log_file"
echo "Results will be saved per-file in '$results_dir/'"
echo "Errors will be saved in '$errors_dir/'"

# Loop through each config file and fetch repos
# All output from this loop goes to the log file
for filename in "${config_files[@]}"; do
    get_repos_for_file "$filename"
done

# Post-processing step: we combine, deduplicate, and filter results
# This part runs after the main loop, output goes to log file
combined_output_file="$parent_dir/combined_unique_results_${today}.txt"
echo "Combining results from '$results_dir/' into '$combined_output_file'..."

# Clear or create the combined file
> "$combined_output_file"

# Find all result files, concatenate them, filter only URLs, sort uniquely, and save
find "$results_dir" -name "*_results_${today}.txt" -type f -print0 | xargs -0 cat | grep '^http' | sort -u
↪  > "$combined_output_file"

if [[ $? -eq 0 ]]; then
    echo "Successfully combined, deduplicated, and filtered results into '$combined_output_file'."
    # Optional: Count unique URLs
    unique_count=$(wc -l < "$combined_output_file")
    echo "Total unique repository URLs found: $unique_count"
else
    echo "Error during combining/deduplicating results. Check individual files in '$results_dir/'."
fi
```

```
316 # End of post-processing
317
318 echo "Log finished at $(date)"
319 echo "All searches complete."
320
321 exit 0
```

Fragmento de codigo 1: `src/get_urls.sh`

Anexo B

Aplicaciones de muestra del dataset

En este anexo se presentan ejemplos representativos de las aplicaciones recolectadas en el dataset. Para cada muestra, se detalla su propósito, una imagen de referencia, y un fragmento del archivo de configuración *Serverless Framework* utilizado.

Muestra 1: MapItOut

Propósito: MapitOut es una herramienta que calcula la distancia que se puede recorrer con ciertos medios de transporte dentro de un tiempo de viaje específico dentro del área de Ámsterdam. También muestra puntos de interés como centros educativos.

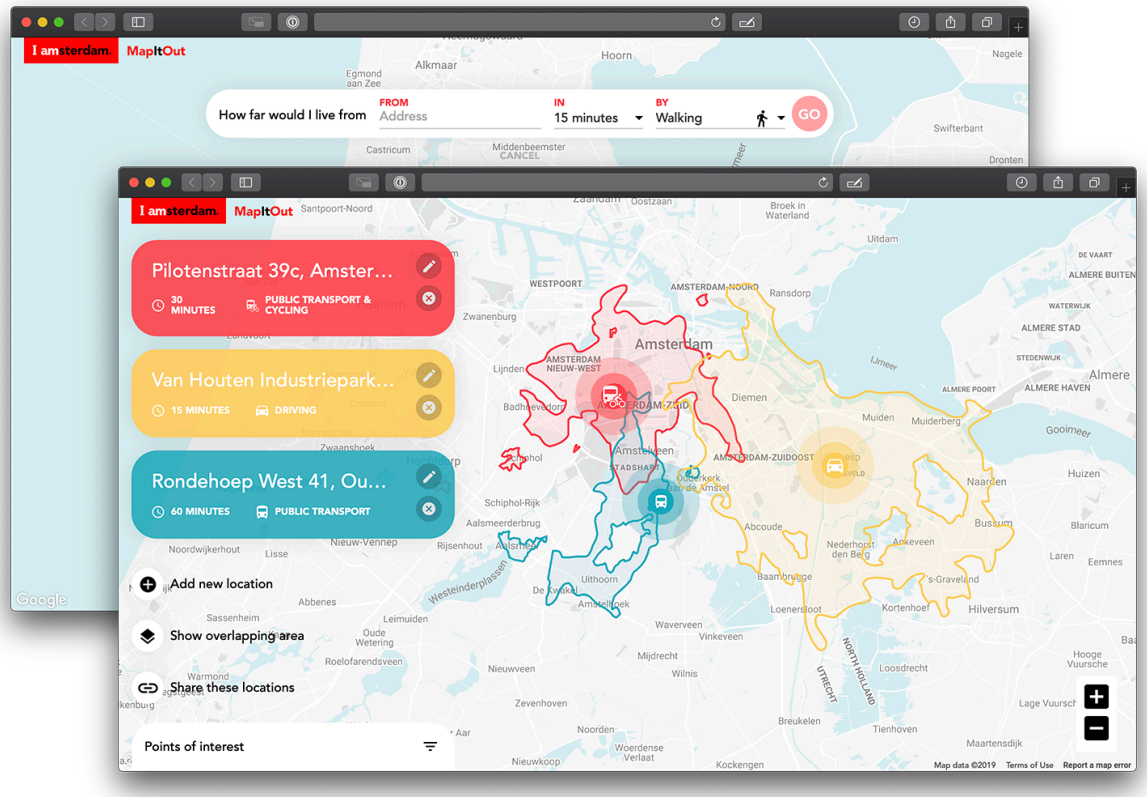

Figura 1: Vista previa de la aplicación.

Fragmento de archivo `serverless.yml`:

```yaml
service: mapitout-proxy

plugins:
  - serverless-offline
  - serverless-prune-plugin
  - serverless-s3-sync
  - serverless-cloudfront-invalidate
```

```yaml
provider:
  name: aws
  runtime: nodejs16.x
  stage: production
  region: eu-west-1
  environment:
    TRAVELTIME_APP_ID: ${env:TRAVELTIME_APP_ID}
    TRAVELTIME_APP_KEY: ${env:TRAVELTIME_APP_KEY}

functions:
  time-map:
    handler: functions/time-map.handler
    events:
      - http:
          path: time-map
          method: post
          cors: true
```

Muestra 2: AWS Billing to Slack

Propósito: Esta aplicación envía desgloses diarios de los costos de AWS a un canal de Slack.

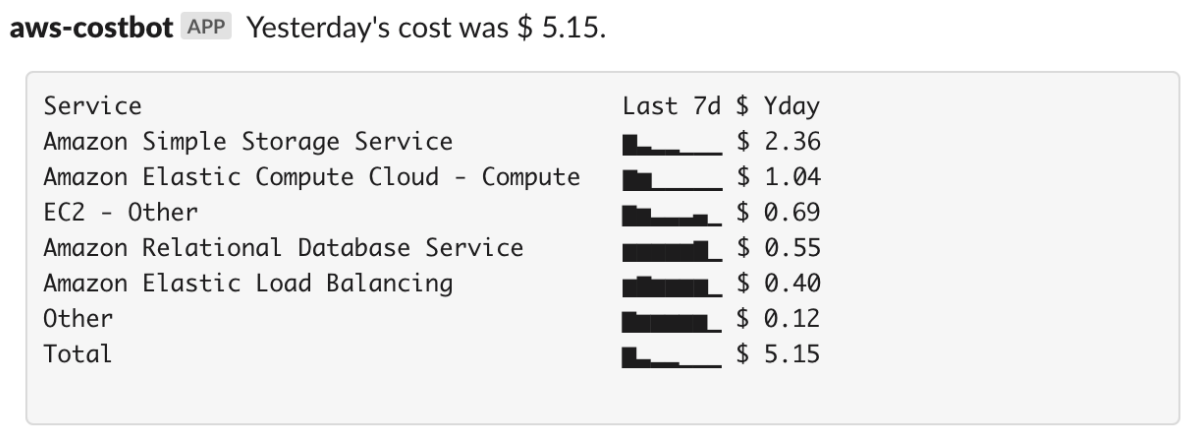

Figura 2: Vista previa de la aplicación.

Fragmento de archivo `serverless.yml`:

```yaml
service: aws-billing-to-slack

plugins:
  - serverless-python-requirements

provider:
  name: aws
  runtime: python3.10

  iam:
    role:
      statements:
        - Effect: "Allow"
          Action:
            - "ce:GetCostAndUsage"
          Resource: "*"
          # Needed to get account alias
        - Effect: "Allow"
          Action:
            - "iam:ListAccountAliases"
          Resource: "*"

functions:
  report_cost:
    handler: handler.lambda_handler

    # Keep costs minimal
    memorySize: 128
    timeout: 10

    events:
      - schedule: cron(0 15 * * ? *)
```

Muestra 3: paraphraser-api

Propósito: Backend para herramienta de parafraseo que aprovecha las API de LLMs específicos (ChatGPT, Gemini). Implementado como una aplicación AWS Lambda y utiliza Serverless Framework para su implementación.

Fragmento de archivo `serverless.yml`:

```yaml
service: paraphraser-api

provider:
  name: aws
  runtime: provided.al2
  architecture: arm64
  deploymentMethod: direct

functions:
  paraphrase:

    handler: bootstrap
    timeout: 20
    memorySize: 128
    events:
      - httpApi:
```

```
          method: post
          path: /paraphrase
```

Muestra 4: mes-driver-service (DVSA)

Propósito: Microservicio encargado de la obtención y gestión de información de licencias de conducir para candidatos a exámenes de manejo en el Reino Unido.

Fragmento de archivo `serverless.yml`:

```
service: mes-driver-service

provider:
  name: aws
  runtime: nodejs20.x

functions:
  getDriverPhotograph:
    handler: src/functions/getDriverPhotograph/framework/handler.handler
    events:
      - http:
          path: 'driver/photograph/{drivingLicenceNumber?}'
          method: get
  getDriverSignature:
    handler: src/functions/getDriverSignature/framework/handler.handler
    events:
      - http:
          path: 'driver/signature/{drivingLicenceNumber?}'
          method: get
  postStandardDriver:
    handler: src/functions/postStandardDriver/framework/handler.handler
    events:
      - http:
          path: 'driver/standard'
          method: post

plugins:
  - serverless-dotenv-plugin
  - serverless-webpack
  - serverless-offline
```