Arquitectura de microservicios distribuidos para una plataforma que orquesta actividades orientadas a la recolección de datos con intervención humana

Jose Arcidiacono*[1][000-0003-0300-5213], Patricia Bazán [4][0000-0001-6720-345X], Alejandra B. Lliteras[2,3] [0000-0002-4148-1299]
[1] UNLP, Facultad de Informática, La Plata, Buenos Aires, Argentina
josefine2_94@hotmail.com
[2] UNLP, Facultad de Informática, LIFIA, La Plata, Buenos Aires, Argentina
[3] CICPBA, Buenos Aires, Argentina
alejandra.lliteras@lifia.info.unlp.edu.ar
[4] UNLP, Facultad de Informática, LINTI, La Plata, Buenos Aires, Argentina
pbaz@info.unlp.edu.ar

Docentes tutores: Patricia Bazán, Alejandra B. Lliteras

Categoría: Final de Grado – Carrera: Licenciatura en Sistemas

Resumen. Este trabajo presenta la arquitectura distribuida y basada en microservicios, definida para dar soporte a una plataforma para generar y reusar workflows de actividades de recolección de datos, que involucren la intervención humana, y cuya ejecución se realiza desde una aplicación móvil. Tanto la arquitectura como la plataforma conforman la solución propuesta como Tesina de Grado realizada y dirigida por las autoras de este trabajo, siendo el foco principal únicamente la arquitectura sin abordar detalles de la plataforma. Se presentan también las características principales de la recolección de datos con intervención humana.

Keywords: Sistemas Distribuidos. Workflow. Microservicios. Recolección de Datos. Arquitectura

1 Introducción

Los sistemas distribuidos pueden concebirse como aquellos cuya funcionalidad se encuentra fraccionada en componentes que al trabajar sincronizada y coordinadamente otorgan la visión de un sistema único, siendo la distribución, transparente para quien hace uso del sistema [1]. En términos computacionales, se dice que un sistema distribuido es aquel cuyos componentes, de hardware o de software, se alojan en nodos de una red comunicando y coordinando sus acciones a través del envío de mensajes [2].

El concepto de componente se concibe como pieza funcional autónoma y con interfaces bien definidas alcanza al área del desarrollo de software, pero también puede representar la idea de una pieza que, cuando se combina con los demás componentes,

forma parte de un todo. En términos estrictamente informáticos, un componente es cada parte modular de un sistema de software.

Las principales razones para la distribución son: 1- Funcionales: cada componente posee capacidades funcionales diferentes, pero coordinan sus acciones con un objetivo común, 2 - Inherente al dominio: se realiza dentro de un mismo dominio de aplicación y cuyas componentes se identifican y modelan dentro de este dominio y 3- Balanceo de carga y distribución: se realiza para poder asignar tareas a distintos procesadores a fin de mejorar el rendimiento general del sistema.

Los sistemas distribuidos concebidos como aquellos que funcionan como si se tratara de un sistema único, no hacen ninguna presuposición acerca de la variedad funcional de sus componentes ni de cómo estos componentes se comunican. De esta manera las variantes que existen han dado origen a sistemas distribuidos de distinta índole y que van de la no-distribución (sistemas monolíticos) hacia la distribución completa (microservicios orquestados y levemente acoplados).

Un sistema monolítico es aquel que se concibe como un único elemento funcional donde sus prestaciones se ofrecen a través de una sola pieza de código que resuelve tanto la presentación al usuario (interface), como el acceso a los datos (trata con operaciones de entrada/salida) y a su vez resuelve la lógica algorítmica del problema que aborda. Estos sistemas de información se construían en un único lenguaje de programación, sobre un único computador y con un único sistema operativo como plataforma. La comunicación con el usuario y también con el programador, era a través de terminales de caracteres que responden únicamente a comandos lanzados por consola. Su ubicación en el tiempo se remonta a los años ‘70.

Una versión mejorada de los sistemas de información así concebidos, pudo aportarse con las técnicas de programación estructurada, los lenguajes de programación y las metodologías de programación modular introducidas por Edsger Dijkstra [3]. En este sentido, los sistemas de información comenzaron a ser concebidos como un conjunto de componentes o niveles de complejidad, aunque permanecían ejecutándose en una misma computadora y por lo tanto podemos considerarlos no-distribuidos.

Entre los objetivos que persiguen los sistemas distribuidos y que fueron tenidos en cuenta a la hora de plantear la arquitectura propuesta encontramos:

- Apertura. Un sistema distribuido es abierto cuando ofrece extensibilidad y mantenibilidad mediante la adhesión a estándares que describen la sintaxis y semántica de los servicios que ofrece.
- Escalabilidad. Un sistema distribuido es escalable cuando ofrece capacidad de crecimiento en términos de cantidad y tipo de recursos que administra y cantidad usuarios que lo acceden.
- Accesibilidad. Un sistema distribuido es accesible cuando garantiza el acceso a los recursos distribuidos que lo componen a todos los usuarios y en todo momento.
- Transparencia. Un sistema distribuido es transparente en la medida que oculta su heterogeneidad y su distribución a los usuarios, quienes conservan la visión de un sistema único.

Como consecuencia de perseguir estos objetivos, los sistemas distribuidos se enfrentan a los siguientes desafíos.

- Concurrencia. Los recursos de un sistema distribuido se caracterizan por estar expuestos al acceso concurrente de uno o más usuarios, debiendo ser capaz de planificar los accesos evitando *deadlocks*.
- Seguridad. Además de garantizar la seguridad a nivel de transporte de datos, debido al alto grado de promiscuidad de la red como medio de comunicación, un sistema distribuido debe abordar la seguridad en términos de autenticación de usuarios, permisos de acceso sobre los recursos y auditoría del uso de los mismos.
- Heterogeneidad. Un sistema distribuido es heterogéneo en cuanto es capaz de permitir que los usuarios accedan a servicios y ejecuten aplicaciones bajo cualquier plataforma (esto es diferentes redes, hardware, sistemas operativos y lenguajes de programación)

A la luz de la evolución de los sistemas informáticos hacia sistemas distribuidos con microservicios débilmente acoplados, y considerando los objetivos y desafíos que éstos presentan, es que se propone en este trabajo una arquitectura distribuida sustentada en microservicios, como solución para dar soporte a una plataforma mediante la cual sea posible definir y ejecutar actividades orientadas a la recolección y análisis de datos que requieran intervención humana.

El desarrollo de esta plataforma, que no está en el foco de este trabajo, permitirá a usuarios finales, diseñar y crear workflows de manera dinámica que puedan ser ejecutados tanto en tecnología móvil como web, y que contribuyan a actividades de recolección y análisis de datos, con intervención humana.

El trabajo se organiza de la siguiente manera: en la Sección 2 se presentan los fundamentos conceptuales de una arquitectura distribuida con microservicios. La Sección 3 presenta los posibles mecanismos para definir y ejecutar actividades de recolección de datos con intervención humana. En la Sección 4 se analiza el estado del arte y los trabajos relacionados, como antecedente a la arquitectura propuesta, presentada en la Sección 5. Finalmente, en la Sección 6 se abordan los resultados y conclusiones.

2 Arquitectura distribuida con microservicios: fundamentos y evolución

La arquitectura de un sistema distribuido define la estructura completa de un sistema según sus componentes específicos, determinado el rol y funciones que cumple cada componente, la ubicación dentro de la red y la interrelación entre los componentes, lo que determina su rol o interfaz de comunicación.

Las funciones de los componentes son: procesos servidores (atienden requerimientos solicitados por otros procesos), procesos clientes (inician la comunicación, solicitan el requerimiento y esperan la respuesta) y procesos pares (procesos que cooperan y comunican simétricamente para realizar una tarea).

La estructura de software es expresada por niveles de servicio que facilitan la interacción en una misma computadora o en varias de ellas, ofreciendo cada ellos distintos niveles de abstracción [2], tal como se observa en la Figura 1.

En este sentido, se puede decir que una de las variantes a este modelo arquitectónico se sustenta en la manera en que se define el middleware. Dichas variantes muestran la evolución de los sistemas distribuidos desde el clásico cliente/servidor en 2 capas, hasta las arquitecturas orientadas a servicios (SOA - *Services Oriented Architecture*), pasando por las soluciones intermedias de arquitecturas cliente/servidor de 3 capas, siendo la tecnología Web uno de tales ejemplos [1].

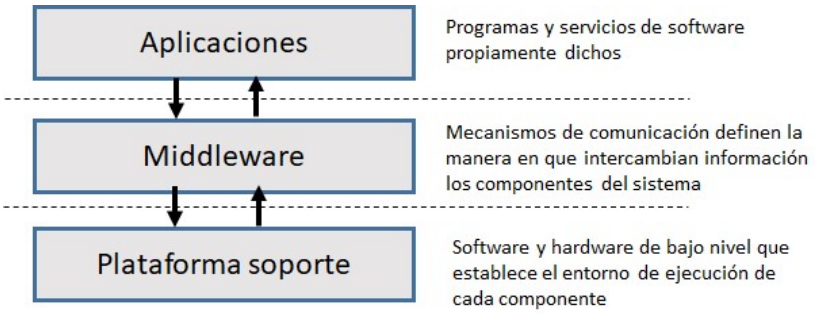

Fig. 1 - Niveles de arquitecturas distribuidas

2.1 De los objetos distribuidos a los servicios

Como antecedente previo a las arquitecturas orientadas a servicios, se encuentran los objetos distribuidos, surgidos desde los conceptos de la orientación a objetos llevados a entornos no locales.

Se puede decir que los objetos distribuidos han sido uno de los antecedentes conceptuales más importantes para los servicios, considerando que sus principales diferencias radican en: 1- los objetos se definen por sus métodos mientras que los servicios ofrecen meta-información para ser utilizados, 2 - en un servicio no existen operaciones, por lo tanto, hay menos cohesión entre ellos.

Un servicio difiere de un objeto o un procedimiento porque se define en función de los mensajes que intercambia con otros servicios; es un elemento que se comprende en términos de la utilidad que brinda, por lo tanto, no puede apartarse del negocio o problema para el cual debe ser útil.

Los servicios son tareas computacionales débilmente acopladas que se comunican vía una red y, al ser provistos por servidores, requieren de una descripción que pueda ser accedida potenciales clientes [4].

Las arquitecturas orientadas a servicios persiguen como objetivos: 1 - La reutilización de componentes tanto desde los aspectos funcionales (evitando duplicación) como mecanismo para favorecer su autonomía, 2 - La interoperabilidad entre aplicaciones y tecnologías heterogéneas y 3 - La flexibilidad para componer, integrar y escalar soluciones.

Para poder alcanzar estos objetivos, una arquitectura orientada a servicios debe incluir los siguientes conceptos: 1 - Servicios, como piezas funcionales que resuelven

un aspecto del negocio, 2 - Alojamiento de los Servicios, o infraestructura habilitante como la operación en un contexto distribuido y 2 - Bajo acoplamiento, o reducción de las dependencias entre servicios.

2.2 Implementaciones con servicios Web

Los Servicios Web (inglés *Web Services*) surgieron en forma paralela a la idea de SOA. La plataforma SOA ordenó su uso sistemático combinando protocolos, perfiles, especificaciones y estándares [5].

Un Web Services es un programa cuyo objetivo es intercambiar información entre dos máquinas, por lo tanto, debe contar con un protocolo de comunicación, según el cual se definen las dos implementaciones posibles para los Web Services: SOAP (*Simple Object Access Protocol*) y REST (*REpresentational State Transfer*). Como su mismo nombre lo indica, el protocolo REST se circunscribe a una transferencia de estado y utiliza siempre el protocolo HTTP (*HyperText Transfer Protocol*) para resolver la transferencia, con la particularidad que desencadena la ejecución de código en lugar de retornar una página HTML. Cuando se menciona RESTFul o API REST, se hace referencia a los programas basados en REST.

Las implementaciones basadas en SOAP son menos flexibles dado que requieren un lenguaje particular de especificación de interfaces o *endpoints*, que debe estar presente necesariamente en el mensaje enviado. Por otra parte, soporta como protocolo de comunicación a HTTP y también otros más como JMS (*Java Message Services*) o FTP (*File Transfer Protocol*), entre otros.

En definitiva, si observamos la evolución de los sistemas distribuidos y retomamos el concepto de servidor de aplicaciones existentes en muchas soluciones tecnológicas, por ejemplo, JavaEE, JBoss o Tomcat, por mencionar algunos, observamos que cualquiera de ellos manejan gran parte de las interacciones entre la capa cliente y la capa de persistencia (esto en un esquema 3 capas clásico) e incluyen servicios de middleware como soporte para web services, manejo transaccional, sistema de mensajes y balanceo de carga, entre otros.

3 La actividad de recolección de datos realizada con intervención humana

A lo largo del tiempo, las personas han empleado diferentes herramientas para recolectar datos. Por ejemplo, documentos estructurados en papel, formularios online y aplicaciones móviles realizadas específicamente para ese fin [6].

Un aspecto relevante de usar tecnología móvil para la recolección de datos, es que, a los sumados por el ser humano, se pueden agregar los obtenidos mediante sensores (como, por ejemplo, la posición GPS obtenida desde el dispositivo) [7].

La recolección de datos usando dispositivos móviles, adicionalmente, puede implicar diferentes estrategias, por un lado, los datos a recolectar, podrían ser tomados en cualquier momento y en cualquier lugar. Un ejemplo de esto, se da cuando un obser-

vador de aves, informa de una especie avistada, donde podría brindar datos como una foto del ave, el lugar dónde la visualiza y la hora [8]. Por otro lado, la recolección de datos podría requerir que esta actividad se realice en un lugar específico, por ejemplo, cuando se quiere conocer todas las especies de árboles de un lugar determinado como podría ser una reserva natural específica [9].

La recolección de datos asistida por tecnología móvil, ha sido usada a lo largo del tiempo en diferentes dominios. Por ejemplo, en [10. "*The inaturalist species classification and detection dataset*"] los autores proponen el uso de la aplicación iNaturalist para el dominio de la biodiversidad, por otro lado, en [11. "*Using smartphones to collect time–activity data for long-term personal-level air pollution exposure assessment*"] los autores proponen una aplicación para la recolección de datos para el análisis de contaminación del aire, mientras que en [12. "*Identification of COVID-19 can be quicker through artificial intelligence framework using a mobile phone–based survey when cities and towns are under quarantine*"] los autores proponen la recolección de datos para analizar aspectos de salud relacionados al COVID-19. Además, en [13. "*AppEAR: Una aplicación móvil de ciencia ciudadana para mapear la ca-lidad de los hábitats acuáticos continentales*"] el autor propone una aplicación para recolectar datos relacionados a ambientes acuáticos, en [14. "*GeoVin: Ciencia Ciudadana para aprender de las vinchucas de Argentina*"] los autores presentan "GeoVIN" una aplicación para recolectar datos ayudando a la sociedad a reconocer vinchucas, y en [15. "*ZIKATracker: A mobile App for reporting cases of ZIKV worldwide*"] los autores proponen una aplicación para recolectar datos del virus Zika en el mundo.

Las aplicaciones que se usan para realizar la actividad de recolección de datos, son generalmente soportadas por arquitecturas distribuidas. En particular, el foco del presente trabajo consiste en analizar arquitecturas existentes de aplicaciones de este tipo, para luego proponer una arquitectura basada en microservicios.

4 Estado del arte y trabajos relacionados

Las herramientas de recolección de datos constituyen sin dudas un ámbito ya desarrollado y existen soluciones previas que fueron consideradas a la hora de validar la solución propuesta en este trabajo. En particular, en este trabajo, se hace foco en la arquitectura distribuida de algunas aplicaciones móviles que permiten ser configuradas para ser utilizadas por usuarios finales y en diferentes dominios.

- **Epicollect.** Epicollect5 es una aplicación móvil y web para la recolección de datos gratuita que proporciona una aplicación web y móvil para la generación de formularios (cuestionarios) y ofrece sitios web de proyectos alojados libremente para la recopilación de datos. Los datos se recopilan utilizando múltiples dispositivos y pueden visualizarse en un servidor central (a través de mapa, tablas, gráficos). La herramienta ha evolucionado y se han generado distintas versiones. En cada caso también ha cambiado la arquitectura.
  En su primera versión Epicollect [16], en 2009, presenta una arquitectura de tipo cliente-servidor, compuesta por un servidor Web (spatialepidemiology.net), accesible tanto por un cliente Web como por una aplicación móvil. El componente spa-

tialepidemiology.net constituye un espacio de acceso público para mostrar y analizar datos epidemiológicos de enfermedades infecciosas utilizando Google Maps and Google Earth. Los científicos o generadores de contenido crean sus formularios desde la aplicación web, que luego se descarga en app móvil y que puede ser utilizada por los agentes de campo para recolectar información que luego se envía al servidor Web.

De manera similar a Epicollect, Epicollect+[17], presentado en 2014, cuenta con una arquitectura de tipo cliente/servidor. Sus componentes coinciden con la versión anterior con la particularidad que se reemplaza el servidor Web por un servidor genérico, manteniendo el mismo mecanismo de interacción.

En 2017, aparece Epicollect5 compuesto por una app móvil y un servidor web (five.epicollect.net). La app móvil descarga formularios desde el sitio web y envía los resultados al servidor. Ofrece un componente API REST para que un cliente externo consulte los datos recolectados, tal como se muestra en la Figura 2.

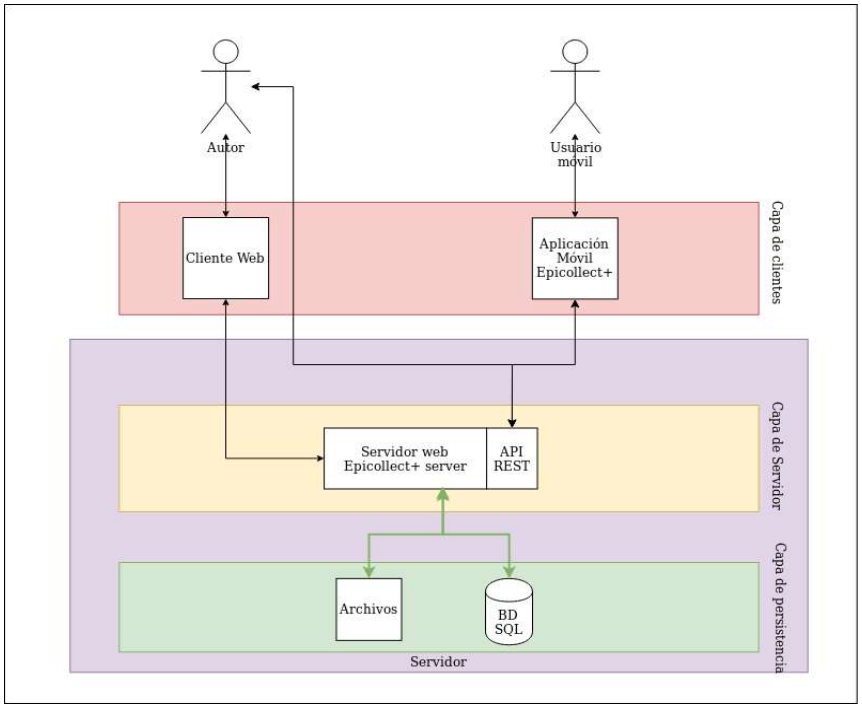

Fig. 2. Arquitectura Cliente/Servidor Epicollect5

- **Zooniverse.** Es un portal de ciencia ciudadana que permite a los usuarios acceder a datos reales para la investigación científica. En sus orígenes se aplicó al dominio de la clasificación de galaxias para luego extenderse a otras ramas de la ciencia.

  Zooniverse[18] presenta una arquitectura de tipo cliente-servidor, donde hay dos componentes servidores: un servidor Web y un API Rest, cada uno accedido por sus respectivos clientes. Además, cuenta con una app móvil que también se comu-

nica con la API, siendo esta pública. Desde el cliente web se puede definir actividades, ejecutarlas y ver los resultados. Con la app móvil solamente se ejecutan actividades y desde el cliente API se puede modificar la definición de actividades y obtener resultados. La Figura 3 muestra la arquitectura de Zooniverse.

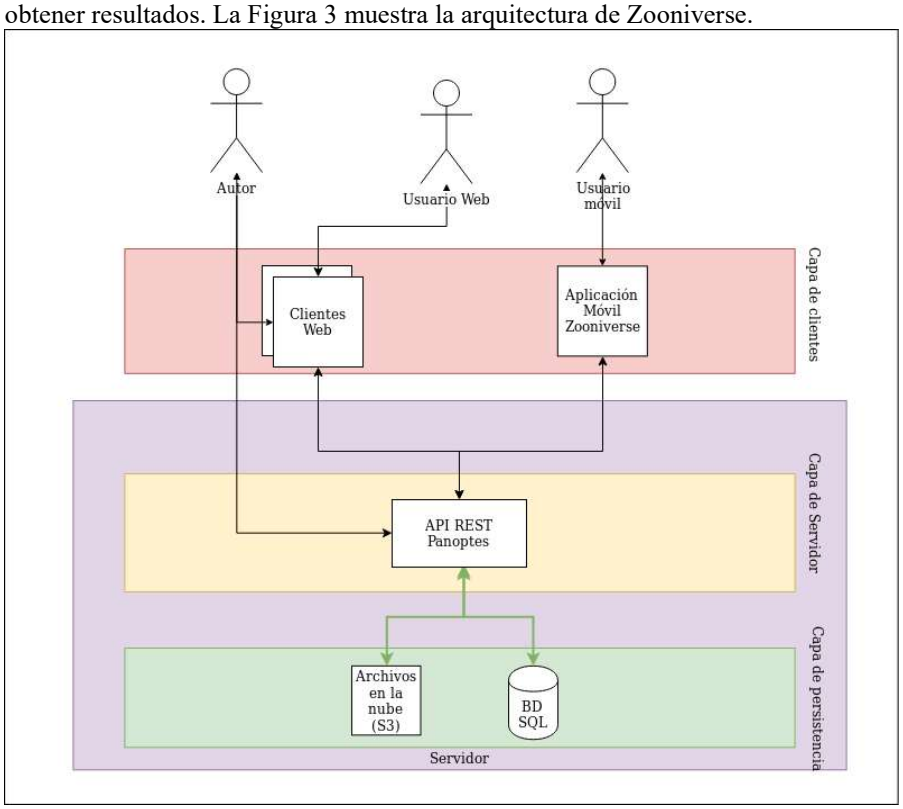

Fig. 3. Arquitectura Cliente/Servidor Zooniverse

- **Open Data Kit.** ODK es un sistema distribuido[1]. Está compuesto por ODK Build y ODK XLSForm, herramientas para diseñar los formularios; ODK Aggregate y ODK Central, opciones de servidor para concentrar los formularios y los datos recibidos; ODK Collect, una app móvil capaz de descargar formularios tanto desde un servidor como desde una planilla de cálculo de Google; y ODK Briefcase, una app de escritorio que permite consultar los datos de Aggregate/Central.
  A diferencia de las otras soluciones, ODK es una herramienta de propósito general, es decir, no está orientada a la recopilación de datos de un dominio específico [19]. Su arquitectura está distribuida en componentes específicas de acuerdo a su objetivo funcional (Figura 4). Los componentes servidores (ODK Aggregate y ODK

1 https://opendatakit.org/

Central) accesibles por clientes de escritorio, clientes API, navegadores y planillas de cálculo.

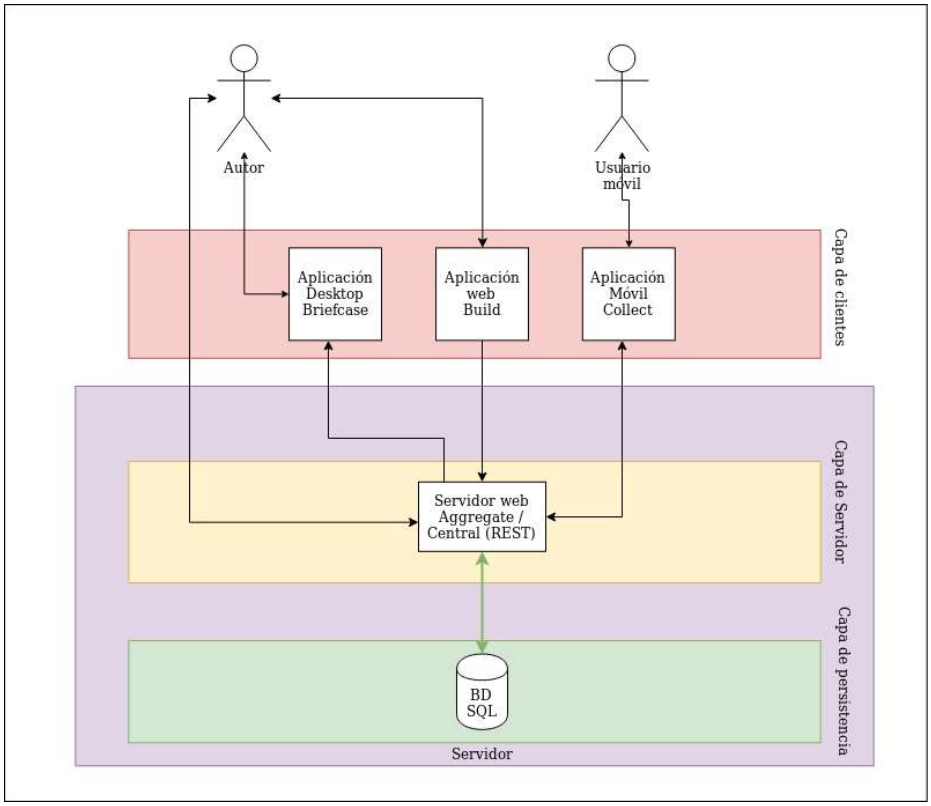

Fig. 4. Arquitectura por Componentes Funcionales ODK

Del análisis arquitectónico de las herramientas mencionadas, se puede concluir que las características comparables más relevantes pueden dividirse en: prestaciones de la herramienta y los componentes que la resuelven, por un lado, y las etapas del proceso de recolección de datos que cubren, por otro. Las etapas identificadas son: 1- definición, que incluye la descripción de la tarea de recolección y el flujo con el que se suceden las tareas, 2- ejecución, que incluye la realización del proceso de recolección y 3 - visualización, que incluye la muestra de los resultados.

En relación a prestaciones todas las herramientas son app móviles y cuentan con API públicas, excepto la primera versión de Epicollect. La arquitectura es cliente/servidor en todos los casos, salvo en ODK que es distribuida en componentes.

En relación a las etapas, todas las herramientas realizan la definición en modalidad web y la ejecución como app móvil, salvo Zooniverse que admite la ejecución tanto móvil como web. La visualización es en su mayoría web, salvo Zooniverse que incluye API para visualización y ODK que provee visualización web y desktop.

En la Tabla 1 se resume la comparación descripta en los párrafos anteriores.

Tabla 1. Comparación de herramientas por funcionalidad y etapas

Funcionalidad	Epicollect	Epicollect+	Epicollect5	Zooniverse	ODK
App Móvil	SI	SI	SI	SI	SI
Definición on line	SI	SI	SI	SI	SI
API Pública	NO	SI	SI	SI	SI
Arquitectura	Cliente/Servidor	Cliente/Servidor	Cliente/Servidor	Cliente/Servidor	Distribuida en Componentes
Etapa					
Definición	Web	Web	Web	Web	Web / XLS
Ejecución	App Móvil	App Móvil	App Móvil	App Móvil/Web	App Móvil
Visualización	Web	Web	Web	Web/API	App Desktop/API

5 DEHIA. Arquitectura propuesta

La arquitectura propuesta para la solución es distribuida en componentes, con cuatro capas y con microservicios, tal como se muestra en la Figura 5.

La Capa *de Clientes* (*Clients layer*) aloja las aplicaciones que permiten la interacción entre los usuarios y el resto de la plataforma. Existirán diferentes tipos de usuario: autores, usuario web y usuario móvil. Cada uno de ellos con acciones definidas acordes al perfil. Esta capa cuenta a su vez con dos componentes: 1- *Cliente Web* que es la interfaz que poseen los autores para definir sus actividades. Permite realizar operaciones sobre las actividades, las tareas y las planificaciones (workflows). También permite ejecutar las actividades de manera on line y, eventualmente, en modo colaborativo y 2- *Aplicación Móvil* que permite ejecutar actividades de manera offline según la planificación definida y envía los resultados al servidor.

Luego se presenta una *Capa de Compuertas (Gateways layer)* que incluye a los componentes tecnológicos que redireccionarán las peticiones entre clientes y los servicios propiamente dichos, contenidos en la *Capa de Servicios (Services Layer)*.

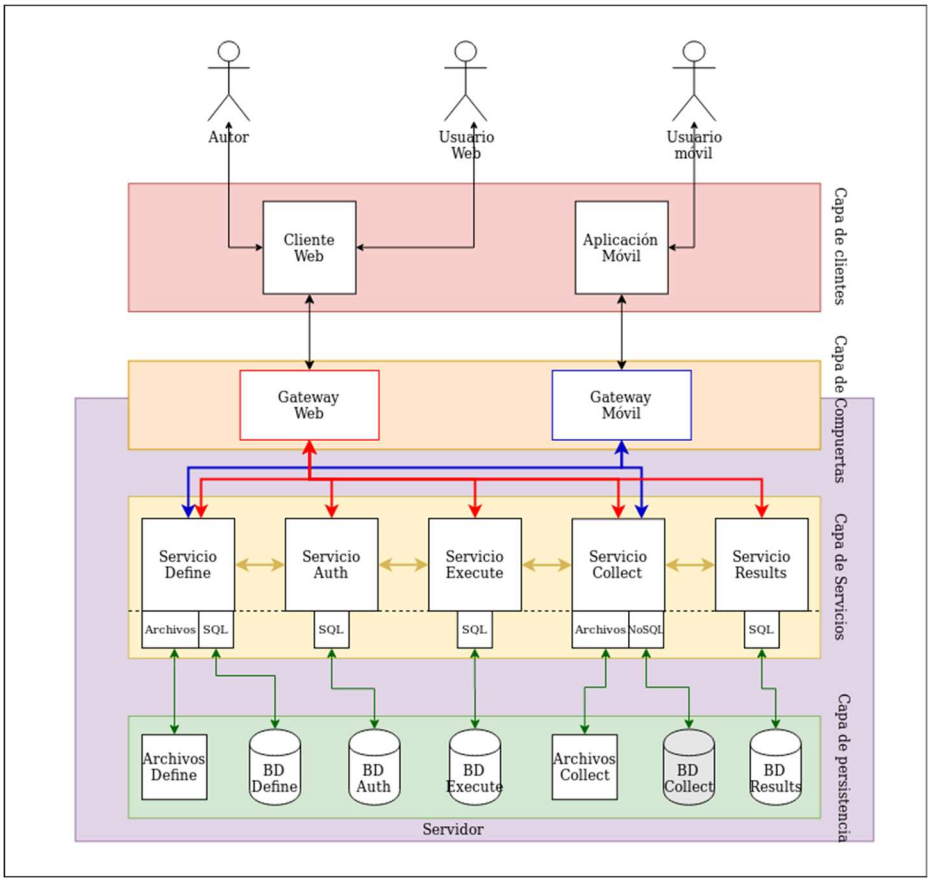

Fig. 5. Arquitectura distribuida por capas y con microservicios de DEHIA

En la Capa de Servicios se cuentan con cinco servicios: 1-*Define*, permite crear actividades, tareas y planificaciones y se comunica con una base de datos y con un sistema de archivos para almacenar imágenes y definiciones, 2- *Auth*, resuelve la autenticación y autorización, brindando tokens de acceso y validando los que le son enviados, 3- *Execute*, realiza las actividades, las cuales toma del servicio *Define*, y lleva el estado de cada instancia de las mismas, 4- *Collect*, concentra las respuestas provenientes de los usuarios y se comunica con una base de datos NoSQL y 5- *Results,* organiza y resume los resultados a partir de las respuestas del servicio *Collect*. Se comunica con un base de datos SQL.

Por último, la *Capa de Persistencia (Persistance Layer)* se ocupa de hacer permanentes los cambios producidos en los datos que gestiona el sistema. Está integrada por archivos y bases de datos, sumando siete componentes: 1 *Archivos Define*, que guarda las imágenes y las definiciones de las actividades del módulo de definiciones, 2- *BD Define* es una base de datos SQL que guarda la información de las actividades, tareas y workflows para poder ser reutilizados, 3- *BD Auth* almacena la información de los usuarios, sus permisos, tokens de acceso, entre otros, también es una base de datos

SQL, 4- *BD Execute* almacena los resultados de las ejecuciones guardando los estados de las actividades en proceso, 5- *BD Results*, guarda las estadísticas e información resumida sobre las respuestas de los usuarios. Ambas son bases de datos SQL, 6- *BD Collec*t es una base de datos NoSQL que almacena las respuestas de los usuarios tal y como son enviadas por la aplicación móvil o el cliente Web y 7- *Archivos Collect* que guardan las imágenes y audios enviados por los usuarios como respuesta.

5.1 Comparación de DEHIA con otras herramientas

Retomando las categorías presentadas en la sección 4, se analiza DEHIA a la luz de las mismas: 1- En relación a la funcionalidad, la arquitectura propuesta cuenta con una aplicación móvil y también una herramienta web para definir las planificaciones on line (workflows). También incluye una API pública y es distribuida con componentes, presentando cuatro capas de servicios diferentes y 2- Respecto de la definición a las etapas cubiertas por la arquitectura, las tres se resuelven en modalidad Web y cuentan con una API. La etapa de ejecución se resuelve, además, con modalidad de app móvil.

La Tabla 2, resume la comparación descripta.

Tabla 2. Componentes de la arquitectura por funcionalidad y etapas cubiertas por DEHIA

Funcionalidad	DEHIA	**Etapa**	DEHIA
App Móvil	SI	Definición	Web/API
Definición on line	SI	Ejecución	App Móvil/Web/API
API Pública	SI	Visualización	Web/API
Arquitectura	Distribuida con componentes por capas		

6 Conclusiones

El trabajo describe la arquitectura propuesta para dar soporte a una plataforma web que permite crear y gestionar workflows para realizar actividades de recolección de datos y de una aplicación móvil que permite ejecutarlos. El foco está puesto en la solución distribuida por componente y con multiservicios que otorga modularidad, escalabilidad y extensibilidad.

El modelo arquitectónico estuvo inspirado en herramientas construidas para la recolección de datos y se enriqueció con una propuesta propia que cumpla los requisitos definidos para la plataforma DEHIA, una herramienta web de autor que permite a usuarios sin conocimientos de programación puedan crear aplicaciones para recolección de datos con intervención humana y que sean ejecutadas en dispositivos móviles.

Además, el foco de la plataforma está en la posibilidad de reusar tareas en diferentes proyectos, así como estructuras de workflows, a los cuales se les podrá configurar nuevas tareas.

DEHIA se encuentra en proceso de desarrollo, pero presenta un grado de avance importante y ha sido sometida a un testeo preliminar con algunos usuarios.

Referencias

1. Bazán P. et al (2017). Arquitecturas, Servicios y Procesos Distribuidos. Una Visión desde la construcción del software. Libro de Cátedra. Editado por EDULP (Editorial de la UNLP). ISBN 978-950-34-1520-7 http://sedici.unlp.edu.ar/bitstream/handle/10915/62354/Documento_completo.pdf-PDFA.pdf?sequence=1
2. Colouris, G et al. (2000) Distributed Systems Concepts and Design 3e. Addison–Wesley. ISBN- 13: 978-0132143011
3. Dahl, O. J., Dijkstra, E. W., & Hoare, C. A. R. (Eds.). (1972). *Structured programming*. Academic Press Ltd.
4. Erl, T. (2007) SOA Principles of Service Design. *Prentice Hall.* ISBN-13: 9780132344821. (25, 119).
5. Pulier, E. et al (2006) Understanding Enterprise SOA. *Manning Publications Co*. ISBN 1-932394-59-1. 2006. (1, 73).
6. Preece, J. (2016). Citizen science: New research challenges for human–computer interaction. *International Journal of Human-Computer Interaction*, *32*(8), 585-612.
7. Kanhere, S. S. (2013, February). Participatory sensing: Crowdsourcing data from mobile smartphones in urban spaces. In *International Conference on Distributed Computing and Internet Technology* (pp. 19-26). Springer, Berlin, Heidelberg.
8. Bonney, R., Cooper, C. B., Dickinson, J., Kelling, S., Phillips, T., Rosenberg, K. V., & Shirk, J. (2009). Citizen science: a developing tool for expanding science knowledge and scientific literacy. BioScience, 59(11), 977-984
9. Brandon, A., Spyreas, G., Molano-Flores, B., Carroll, C., & Ellis, J. (2003). Can volunteers provide reliable data for forest vegetation surveys?. *Natural Areas Journal*, *23*(3), 254-262.
10. Van Horn, G., Mac Aodha, O., Song, Y., Cui, Y., Sun, C., Shepard, A., ... & Belongie, S. (2018). The inaturalist species classification and detection dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8769-8778).
11. Glasgow, M. L., Rudra, C. B., Yoo, E. H., Demirbas, M., Merriman, J., Nayak, P., ... & Mu, L. (2016). Using smartphones to collect time–activity data for long-term personal-level air pollution exposure assessment. *Journal of exposure science & environmental epidemiology*, *26*(4), 356-364.
12. Rao, A. S. S., & Vazquez, J. A. (2020). Identification of COVID-19 can be quicker through artificial intelligence framework using a mobile phone–based survey when cities and towns are under quarantine. *Infection Control & Hospital Epidemiology*, 1-5. 2016/11/21.
13. Cochero, J. (2018). AppEAR: Una aplicación móvil de ciencia ciudadana para mapear la calidad de los hábitats acuáticos continentales.
14. Balsalobre, A., Ceccarelli, S., Cano, M. E., Ferrari, W. A., Cochero, J., & Martí, G. A. (2018). GeoVin: Ciencia Ciudadana para aprender de las vinchucas de Argentina. In II

Congreso Argentino de Ciencia Abierta y Ciudadana (CIACIAR)(Universidad Nacional de San Martín, 2 de noviembre de 2018).

15. Kelvin, A. A., Banner, D., Pamplona, L., Alencar, C., Rubino, S., & Heukelbach, J. (2016). ZIKATracker: A mobile App for reporting cases of ZIKV worldwide. *The Journal of Infection in Developing Countries*, *10*(02), 113-115.
16. Aanensen, David M., et al. "EpiCollect: linking smartphones to web applications for epidemiology, ecology and community data collection." PloS one 4.9 (2009).
17. Aanensen, David M., et al. "EpiCollect+: linking smartphones to web applications for complex data collection projects." *F1000Research* 3 (2014)
18. Simpson, Robert; Page, Kevin R.; De Roure, David. Zooniverse: observing the world's largest citizen science platform. En *Proceedings of the 23rd international conference on world wide web*. 2014. p. 1049-1054
19. Hartung, C., Lerer, A., Anokwa, Y., Tseng, C., Brunette, W., & Borriello, G. (2010, December). Open data kit: tools to build information services for developing regions. In *Proceedings of the 4th ACM/IEEE international conference on information and communication technologies and development* (pp. 1-12).