Una Arquitectura "Serverless" económica para pequeños portales de datos aplicada en un software prototipo para investigación

Axel Wismer, Brenda Elizabeth Meloni, Oscar Carlos Medina
Universidad Tecnológica Nacional – Facultad Regional Córdoba, Cruz Roja Argentina y Maestro López s/n, Ciudad Universitaria, Córdoba, Argentina
axelwismer@gmail.com, {omedina, bmeloni}@frc.utn.edu.ar

Resumen

Los grupos de investigación, a la hora de la puesta en producción de un software prototipo, presentan la problemática de elegir una arquitectura y estimar el costo del despliegue. La elección de una arquitectura para este despliegue, depende del número de proveedores y formas en la que se venden los servicios, especialmente en aplicaciones que deben estar disponibles en internet durante ciertos periodos de tiempo. En este trabajo se presenta una alternativa sobre las tecnologías "serverless" y su implementación en el software prototipo "PatCat" (Pattern Catalogue) que es un catálogo web de Patrones de Negocios aplicados en casos reales de E-Gobierno. La propuesta de este tipo de arquitectura permitió el ahorro de costos y tiempo que se incurren al utilizar servidores que están constantemente activos como los que se utilizan habitualmente en ambientes de producción.

1. Introducción

La puesta en producción de aplicaciones de software es una problemática muy común en los grupos de investigación, y en proyectos pequeños y medianos.

Esto se debe a varios motivos:

- El costo de mantener aplicaciones en producción por periodos extendidos de tiempo puede ser privativo.
- En caso de querer publicar la aplicación solo por cortos periodos de tiempo se debe contar con personal capacitado en DevOps [1] disponible, lo que nuevamente aumenta los costos.
- Se debe sobredimensionar la cantidad de recursos de la aplicación para el evento en que haya un aumento súbdito en la demanda.

Muchos de estos problemas pueden resolverse aplicando una arquitectura serverless [2] como AWS (Amazon Web Services) Lambda [1] [12] la cual crea instancias de la aplicación a demanda del cliente. Estas aplicaciones no deben pagar por los periodos donde no son solicitadas, e incluso pueden permanecer en la capa gratuita de AWS [2] y Azure[3] mientras su uso sea moderado.

Si bien esta arquitectura soluciona los problemas anteriores, en lo que respecta a la ejecución de aplicaciones, esta no contempla la persistencia de los datos.

Existen múltiples opciones respecto a bases de datos en la nube, entre ellas se encuentran también las que son serverless las cuales, al igual que las aplicaciones de dicho nombre, se crean y escalan a demanda.

Sin embargo, durante el desarrollo de pequeños softwares, estas opciones de persistencia pueden resultar excesivas tanto en su costo como en las características que brindan. Por lo que, si el objetivo es desplegar una pequeña aplicación que sea escalable y se mantenga en producción por el menor costo posible, es necesario pensar en nuevas alternativas.

En este trabajo se expone una arquitectura serverless que permite que pequeñas aplicaciones como portales de datos se mantengan en producción de forma gratuita cuando el uso de las mismas es moderado.

2. Elementos de trabajo y metodología

La realización de PatCat (Pattern Catalogue) se enmarca dentro de un trabajo exploratorio de arquitecturas alternativas, motivado principalmente por la necesidad de un catálogo web de patrones de negocio aplicados en casos reales de gobiernos electrónicos.

[1] DevOps: Conjunto de prácticas, herramientas y filosofía cultural que sirve para automatizar e integrar los procesos que comparten el equipo de desarrollo de software y el de TI.

[2] Arquitectura Serverless: Es un modelo de desarrollo y despliegue de aplicaciones en el cual los desarrolladores pueden enfocarse en escribir código sin preocuparse por la gestión directa de servidores o máquinas virtuales subyacentes.

[3] Azure: Plataforma de computación en la nube creado por Microsoft para construir, probar, desplegar y administrar aplicaciones y servicios mediante el uso de sus centros de datos

Inicialmente se definió las diferentes funcionalidades que integran la aplicación, estableciendo tres tiempos de acción para llevar a cabo el desarrollo:

- Análisis y funcionalidades: Se realizaron actividades para definir las fronteras funcionales de los que sería la aplicación y cada una de sus secciones.
- Desarrollo: Fue la programación de cada uno de los módulos
- Correcciones tardías y mantenimiento: Correspondiente a mejoras en performance, seguridad y a nuevas funcionalidades que ocasionalmente pudieran surgir en el desarrollo.

Cada funcionalidad fue dividida en diferentes tareas, para una mejor modularidad y enfoque.

Para cada uno de las acciones previamente mencionadas se estableció un límite de tiempo de un mes y medio para completarlas, dividiéndose este tiempo en:

- Una semana para el análisis
- Cuatro semanas para el desarrollo
- Una semana para los testings integrales

Se eligió Trello [3] [13], como herramienta para gestionar cada una de las tareas, su estado y para el control de los avances, con el objetivo de mantenerse ordenado y siguiendo los plazos establecidos inicialmente.

3. Funcionalidad y Usos

PatCat cuenta con dos vistas diferentes, la del investigador y la del administrador. La primera vista se compone de un buscador donde el investigador puede ingresar datos relacionados con el patrón que está interesado encontrar (ver **Figura 1**). Luego, el buscador explora todos los resultados que tengan coincidencias y los ordena en función del más relevante al menor.

Además, al hacer click sobre un resultado se muestra una vista de detalle donde se pueden ver todos los datos relevantes del padrón. Esta vista contiene enlaces para las descargas de archivos e imágenes que representan la estructura del padrón (ver **Figura 2**).

La vista del administrador utiliza el módulo administración del "framework" Django [4], el cual presenta un estándar para la visualización y modificación de datos, creando las vistas a partir de la definición de los objetos; así y desde esta sección podremos visualizar de manera ordenada los datos alojados en la aplicación, pudiendo establecer filtros según diversos parámetros de la información expuesta, gracias a la flexibilidad de las interfaces expuestas.

Figura 1: Vista del investigador de la aplicación PatCat.

Además, cuenta con funcionalidades de autenticación y manejo de usuarios (ver **Figura 3**), que posibilita una forma ágil y simple de gestionar los ingresos al mencionado portal, permitiendo establecer no solo nuevos usuarios administradores, sino, también manejar grupos con diferentes niveles de acceso y privilegios, garantizando que cada usuario pueda ver solo lo necesario.

En esta vista el administrador puede consultar y modificar todos los elementos de PatCat, estos son:

- Casos de uso: Situación específica en la que ha sido identificado, múltiples patrones pueden estar vinculados a un caso de uso.
- Categorías de caso de uso: Entre las más comunes se encuentran las siguientes:
  - Transferencia de activos
  - Gestión de identidades
  - Votación y gobernanza
  - Seguros
- Contratos Inteligentes
- Papers: artículos científicos publicados que contienen información relacionada a un patrón.
- Patrones

Tipos de patrones

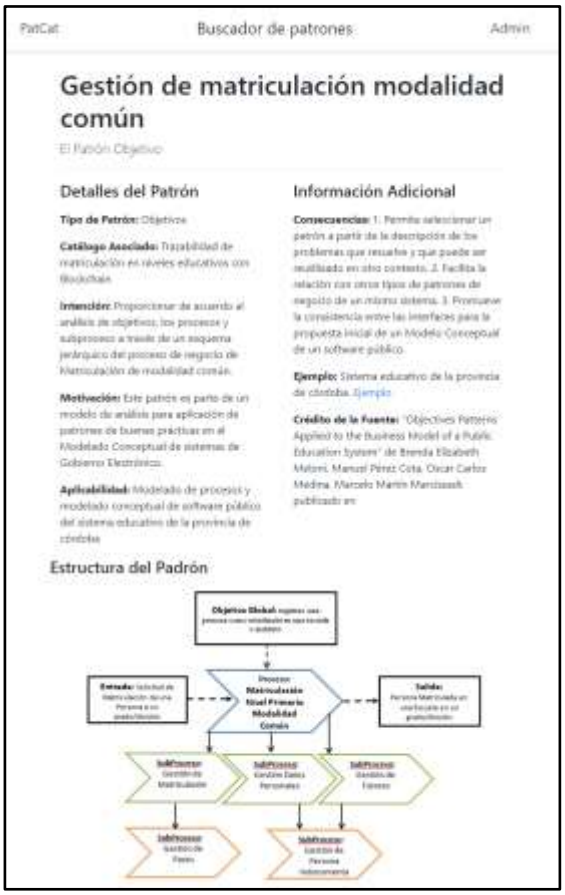

Figura 2: Representación en detalle de un Patrón.

Figura 3: Login del sitio de administración de Django.

Al seleccionar un patrón se muestra un formulario con todos sus elementos donde se puede agregar texto, imágenes y enlaces a archivos como se presenta en la **Figura 4**:

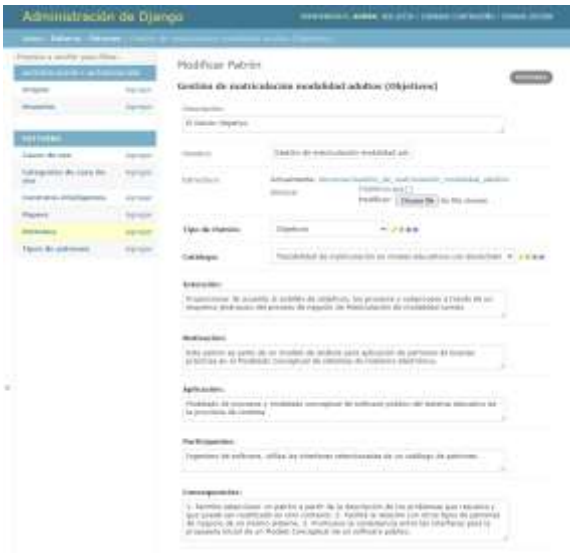

Figura 4: Pantalla principal del sitio de administración de Django.

4. Esquema de Datos

Cada uno de los elementos representados en la vista del administrador corresponde a una tabla en la base de datos (ver **Figura 5**), en el siguiente esquema se ilustran los campos de cada una de ellas. Además, se pueden ver las relaciones de clave foránea establecidas. Este esquema es generado a partir de la definición en Python de los modelos de la base de datos.

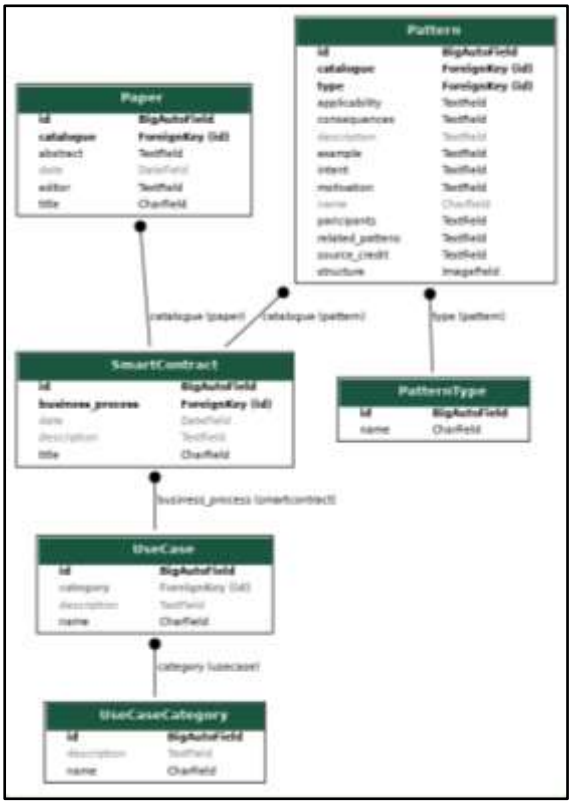

Figura 5: Esquema de la base de datos de PatCat.

5. Arquitectura

En la arquitectura (ver **Figura 6**) de la mayoría de las aplicaciones de software se pueden identificar tres elementos diferentes e igualmente importantes; Interfaz, Lógica y Persistencia de los datos; estos frecuentemente se materializan como diferentes aplicaciones: interfaz gráfica, "backend" y base de datos. Cada una es un servicio que se encuentra constantemente funcionando a la espera de peticiones.

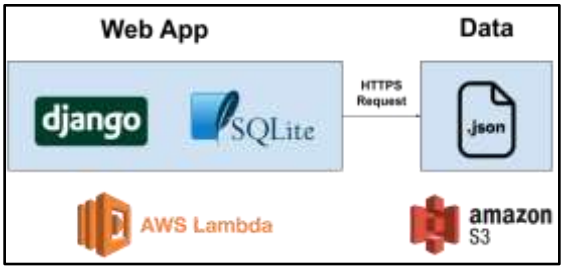

Figura 6: Detalle de la arquitectura de PatCat, evidenciando su integración con Django y AWS.

[4] Bucket S3: Es un contenedor de almacenamiento en la nube que se utiliza para almacenar datos, como archivos, imágenes o datos de aplicaciones

En contraste aquí se propone una nueva arquitectura que se compone de dos servicios (ver **Figura 7**): Una aplicación Django que entrega páginas estáticas pre-renderizadas y una base de datos de archivos, la cual almacena los datos en formato json[4].

El costo de estos dos componentes depende exclusivamente del uso que se les dé y no del tiempo que se encuentran disponibles. Django cuanta además con una base de datos en memoria, la cual permite realizar consultas complejas en la base de datos utilizando un sistema de manejo de base de datos (DBMS).

6. Secuencia de una petición

En la **Figura 7** se presenta el diagrama de secuencia de una petición, es importante recordar que en una arquitectura serverless, el estado de una aplicación no se mantiene entre las distintas peticiones que se le realizan. Por eso en cada una es necesario consultar la base de datos almacenada en el "bucket" de S3[4] y construirla en memoria para poder responder a consultas complejas.

Lo antes mencionado presenta una clara limitación en el sistema dado que el tiempo de construcción de la base de datos en memoria es linealmente dependiente de la cantidad de datos que se desean cargar.

Se debe considerar que el tamaño de la base de datos deberá mantenerse en el orden de los kilobytes para evitar retrasos y tiempos de espera prolongados. Sin embargo, en muchos casos, esta restricción no es significativa considerando el tamaño del sitio que se planea desarrollar.

En el mismo bucket, se pueden almacenar documentos, imágenes y videos sin restricción de dimensión. Finalmente, si se detectan cambios en la base de datos, se generará un archivo. json a partir de los datos en memoria y se lo almacenará en el bucket S3 sobreescribiendo a la anterior base de datos.

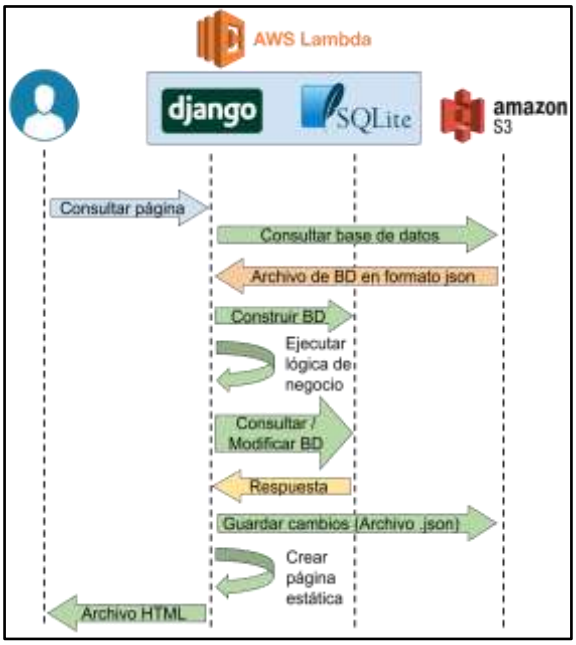

Figura 7: Diagrama de secuencia que representa el proceso que recorre una petición en la aplicación junto a la tecnología interviniente.

Aquí nos encontramos frente a la segunda limitación del sistema, si múltiples administradores se encuentran modificando el mismo objeto en el sitio, solo el último cambio persistirá en la base de datos. Nuevamente, esta restricción no representa un problema dado que, en la mayoría de los casos, sitios de este tamaño no requieren múltiples administradores que realicen cambios de manera concurrente.

7. Elección del framework Django

Siguiendo la filosofía de desarrollar al menor costo posible, se debe considerar no solo el costo monetario, sino el tiempo de desarrollo, testing y despliegue que requerirá la aplicación.

La tecnología AWS Lambda soporta nativamente los siguientes lenguajes de programación: Java, Go, PowerShell, Node. js, C#, Python, y Ruby. Entre los principales candidatos antes mencionados se redujo la elección entre el framework Express Js del lenguaje Javascript y el framework Django de Python. Esta decisión se basó principalmente por ser los frameworks más populares de los lenguajes de mayor uso en el momento. Si bien ambos son apropiados para el desarrollo de grandes sistemas serverless, se optó por el Django por la gran cantidad de funcionalidades que este aporta en sistemas de estas características, las cuales son:

- Sistema de definición de objeto de dominio (Modelos) que se integran con el ORM[5] permitiendo acceso directo a los datos sin necesidad de utilizar SQL[6].
- Sistema De migraciones que permite crear y modificar las tablas de una base de datos, independientemente de la tecnología que se utilice.
- Módulo del administrador que permite generar una interfaz de usuario amigable, la cual incluye menús y formularios, sin necesidad de escribir código adicional.
- Librerías que permiten la conexión con todas las tecnologías de AWS previamente mencionadas.
- Seguridad de manera nativa, Django intenta mitigar las posibles fugas de información que podrían darse. Está diseñado para proteger al sitio y al programador de muchos de los errores de seguridad que se producen en la actualidad, como SQL Injection XSS (Cross Site Scripting), Cross-site request forgery (CSRF), entre otros [5].

Lo antes detallado es un conjunto de vulnerabilidades de seguridad que pueden estar presentes y explotarse en aplicaciones web por atacantes maliciosos, con el fin de comprometer la integridad, confidencialidad o disponibilidad de los datos y la funcionalidad de una aplicación. Es muy importante implementar buenas prácticas de codificación y protegerse contra estas amenazas.

Todas estas características permitieron acelerar notablemente el desarrollo de la aplicación y redujeron notablemente los errores alrededor de la creación y modificación de los datos.

8. Módulo Memory DB

Para la implementación de esta arquitectura se implementó un nuevo módulo llamado Memory DB, el cual gestiona la creación, carga, actualización y persistencia de la base de datos en memoria.

Para garantizar la futura compatibilidad e integración con el framework Django, este módulo hace uso del ORM (modelo de programación que permite mapear las estructuras de una base de datos relacional) [6] del framework y del middleware (framework de Django que funciona como punto de partida en el procesamiento de solicitudes / respuestas de Django. Es un sistema ligero y de bajo nivel para alterar globalmente las entradas y salidas) [7] [14] que permite la integración con la base de datos en memoria y con el bucket S3 de AWS.

Para la carga y creación de los datos en memoria, se utilizan los comandos "*loaddata*" y "*dumpdata*" propios de Django, que permiten serializar y deserializar la base de datos para poder ser guardada como un archivo. json.

El módulo aprovecha el uso de las señales provistas por Django, que permiten vincular acciones programables a eventos específicos del sistema. En este caso se vincula el script que gestiona la serialización y guardado de los datos en S3 al evento de la actualización de un objeto en memoria.

Además, el módulo cuenta con un script de inicialización de la base de datos, el cual se inicializa de manera previa a la ejecución de la lógica de negocio de la aplicación.

9. Despliegue y mantenimiento

Para el despliegue de la aplicación se optó por utilizar la librería Django-zappa [8]. Esta herramienta provee los elementos necesarios para comprimir una aplicación y almacenarla en un bucket de S3, el cual es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento [9] [15]. Permite publicar la misma en la plataforma Cloudwatch, la cual maneja los recursos de AWS. Finalmente, la librería pública la función lambda, verificando que esta se encuentre saludable y asignándole una URL para que pueda ser accesible de forma pública.

Previo al despliegue de la aplicación, se creó un bucket S3; este contiene la base de datos y todos los archivos estáticos (imágenes y hojas de estilo en cascada) necesarios para el correcto funcionamiento de la aplicación.

10. Seguridad

Se implementa un sistema de seguridad basado en usuarios, roles y permisos con el módulo IAM de AWS (ver **Figura 8**). Para esto, se definieron roles que permiten a un usuario crear buckets de S3, construir y desplegar funciones lambda y hacer consultas al bucket de S3 que contiene la base de datos.

Para el acceso de la librería zappa y de la aplicación a este usuario, se creó un token de acceso y se lo configuró en el CLI de AWS, reduciendo el riesgo de exposición de las credenciales de AWS.

Durante la ejecución de la función lambda se registran los eventos o logs en el servicio Cloudwatch [10] para su posterior auditoría.

11. Resultados

La aplicación PatCat se encuentra desplegada de forma ininterrumpida desde el 12 de noviembre de 2022. Durante este periodo la aplicación ha sido actualizada múltiples veces sin necesidad de interrumpir la ejecución de la misma. Además, al mismo tiempo, el software no ha producido ningún gasto al grupo de investigación, dado que se mantiene en la capa gratuita de AWS, la cual permite realizar un millón de consultas por mes, antes de generar algún tipo de facturación.

Figura 8: Esquema de seguridad de PatCat.

Desde su despliegue, la aplicación Django ha recibido una media de 11 mil invocaciones por mes. Cada invocación es una petición HTTP al servidor como lo muestra la **Figura 9**:

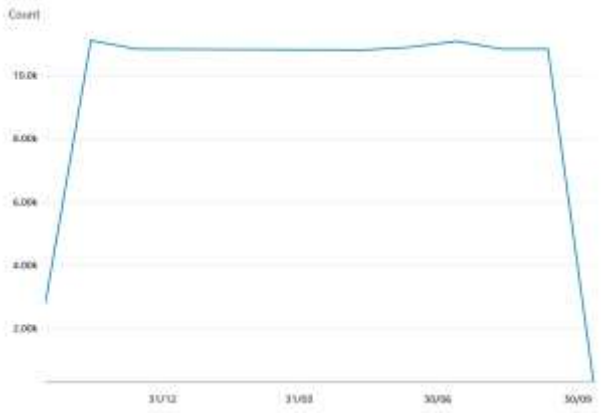

Figura 9: Cantidad de invocaciones mensuales a la función Lambda en un periodo de un año.

Estas métricas han sido obtenidas de la plataforma AWS Cloudwatch. El tiempo de ejecución medio de la función lambda es de 144 ms. graficado en la **Figura 10**:

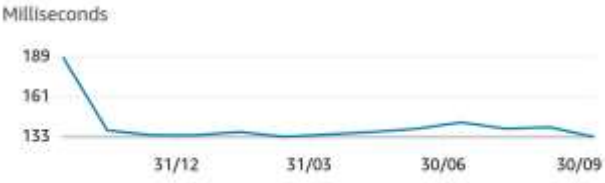

Figura 10: Duración medio mensual de la función lambda de en un periodo de un año.

El almacenamiento de datos en s3 alcanzó un máximo de 1.93 Mb incluyendo la base de datos e imágenes como se puede apreciar en la **Figura 11**:

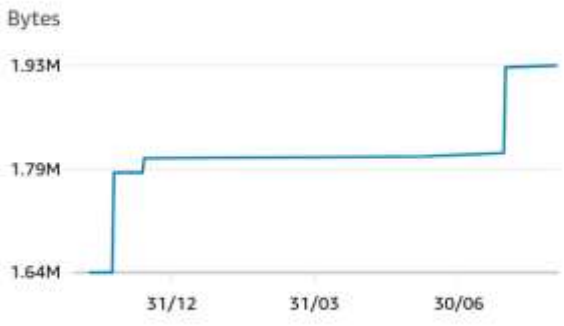

Figura 11: Espacio utilizado en bucket de s3 en el periodo de un año.

La base de datos es un archivo de 39.6 KB de datos. Considerando que la capa gratuita de AWS Lambda es de 1 millón de invocaciones mensuales y 3.2 millones de tiempo de computación, se puede concluir que la aplicación se encuentra ampliamente por debajo del tiempo.

12. Discusión

Una de las características que no se consideran normalmente al momento de elegir una arquitectura es el costo de despliegue de la misma. Si bien este valor puede ser difícil de estimar debido al número de proveedores y formas en la que se venden los servicios, en la actualidad se considera que elegir una arquitectura serverless puede tener grandes ventajas en este aspecto. Esto es cierto especialmente en aplicaciones que deben encontrarse disponibles en internet durante largos periodos de tiempo, pero que por su limitado uso no justifican el costo de despliegue con tecnologías tradicionales, que pueden implicar altos gastos a sus prestadores.

Actualmente se puede ver un avance en la adopción de las tecnologías serverless, a tal nivel que el 40 % de los profesionales de sistemas utilizan arquitecturas serverless en sus aplicaciones, permitiendo ahorro de costos y tiempo que se incurren al utilizar servidores que están constantemente activos. Además, permite ser más amigable con el ambiente, reduciendo así la huella de carbono [5] debido a que solo se utilizan los recursos necesarios, utilizando menos energía en procesos inactivos o poco eficientes [11]

13. Conclusión

Finalizado el desarrollo y puesta en producción de PatCat, junto con la prueba del sistema por parte de usuarios investigadores, y teniendo en cuenta el tiempo que este sistema se ha mantenido en la nube, se puede concluir que la arquitectura expuesta en este paper, ha transformado significativamente la forma en que se realizaron las tareas de investigación y transferencia del conocimiento en el marco de la presente investigación.

Sumado a esto, la implementación serverless, podría llevar a numerosos beneficios importantes dentro de los que podemos destacar:

1. Reducción de costos: Como bien se mencionó en la sección de discusión, el despliegue y la implementación son aspectos muy importantes, y la elección de la tecnología serverless supuso una alternativa eficiente para PatCat, permitiendo que la aplicación estuviese operativa gratis desde su puesta en funcionamiento, manteniéndose dentro de la capa sin costos de AWS.
2. Escalabilidad: Las arquitecturas serverless, son fácilmente escalables según la demanda que sufran, aspecto fundamental para PatCat, ya que cuando el tráfico aumenta, también lo hacen sus recursos, permitiendo una mejor eficiencia al entregar las respuestas solicitadas por los usuarios, en contraposición en periodos de inactividad, se mantiene estable con los mínimos insumos requeridos para así ahorrar en dinero y energía.
3. Seguridad: Transversal a los diferentes servicios cloud utilizados, se ofrecen una amplia gama de herramientas, esto es aprovechado en la aplicación, principalmente para mantener los datos confidenciales securizados y combinado con Django, genera una plataforma con un alto grado de prevención a ataques de usuarios maliciosos.
4. Implementación sencilla: La modalidad serverless permite a los desarrolladores olvidarse de los tediosos esfuerzos de infraestructura, lo que generó que toda la energía del equipo estuvieran 100% enfocada en la codificación y los datos
5. Disponibilidad: Con Lambda provisto por AWS, es posible configurar una aplicación distribuida en diferentes regiones geográficas, esto permitió a PatCat estar disponible rápidamente alrededor del mundo con tiempos de respuesta óptimos, de forma robusta y resistente a fallos que pudieran ocurrir.
6. Multilenguaje: La utilización de Lambda mediante AWS, es soportado por múltiples lenguajes de programación, es por esto que fue útil para PatCat, permitiendo construir la aplicación en Python mediante el uso del framework Django, siendo el lenguaje con el cual el equipo se sentía más cómodo y seguro.

Es por lo antes mencionado, que la tecnología serverless, es fundamental para la construcción de aplicaciones con un tráfico variable en el tiempo, que necesite variar su dimensión según las peticiones que recibe y si se busca una reducción en el tiempo de desarrollo, costos, mantenimiento y de infraestructura permitiendo enfocarse en el código, tareas de proyecto y gestión, obteniendo un rápido despliegue, garantizando disponibilidad, seguridad y robustez.

[5] Huella de Carbono: Cantidad de gases de efecto invernadero liberados a la atmósfera como resultado de las actividades humanas o de un producto.

Agradecimientos

Los autores agradecen a U.T.N., por medio del Centro CIDS y a CYTED, por medio de la Red RIBCi, por el apoyo financiero que hizo posible la realización de este proyecto de investigación y la vinculación internacional de su equipo de investigadores respectivamente.

Referencias

[1] ¿Qué es AWS Lambda? Extraído del sitio: https://docs.aws.amazon.com/es_es/lambda/latest/dg/welcome.html

[2] Capa gratuita de AWS. Extraído del sitio: https://aws.amazon.com/es/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all

[3] ¿Qué es Trello? Extraído del sitio https://trello.com/es/tour

[4] Django Framework. Extraído del sitio https://www.djangoproject.com/

[5] The Django Admin Site. Extraído del sitio https://docs.djangoproject.com/en/4.2/ref/contrib/admin/

[6] ¿Qué es un ORM? Extraído desde https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html

[7] Middleware in Django. Extraído de https://docs.djangoproject.com/en/4.2/topics/http/middleware/

[8] Zappa - Serverless Python. Extraído de: https://github.com/zappa/Zappa

[9] ¿Qué es Amazon S3? Extraído de https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/Welcome.html

[10] ¿Qué es Amazon CloudWatch? Extraído de https://docs.aws.amazon.com/es_es/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html

[11] La tecnología que ayudará a salvar el mundo: ahorro de energía y “serverless”. Extraído de: https://prensariotila.com/la-tecnologia-que-ayudara-a-salvar-el-mundo-ahorro-de-energia-y-serverless/

[12] John Chapin & Mike Roberts, “Programming AWS Lambda, Build and Deploy Serverless Applications with Java”, O´REILLY 2020.

[13] Pedro Lamban Castillo, “Gestión de proyectos con Trello”, Libro-e 2019.

[14] Ben Shaw, Saurabj Badhwar, Andrew Bird, Bharath Chandra KS, Chris Guest, “Web Develoment with Django: A definitive guide to building moren Python web applications using Django 4”, 2nd. Edition Kingle 2021.

[15] Luis Joyanes Aguilar, “Computacion en la nube”, 2da Edición Marcombo 2022.