ReTEC

Revista Electrónica de Tecnología, Educación y Ciencia
ISSN: 2953-5654
http://retec.unsa.edu.ar
Universidad Nacional de Salta

Desarrollo de sistemas de microservicios, y evaluación de modelos arquitecturales y mecanismos de seguridad en relación al rendimiento

Sebastián José García Fontana, Nelson Rodríguez

Departamento de Informática, Facultad de Ciencias Exactas Físicas y Naturales
Universidad Nacional de San Juan
sebasgarciafontana@gmail.com , nelson@iinfo.unsj.edu.ar

Revista Electrónica de Tecnología, Educación y Ciencia,

Volumen 1, Número 3, pág. 21-31, jun, 2026. ISSN: 2953-5654

Disponible en http://retec.unsa.edu.ar/

Desarrollo de sistemas de microservicios, y evaluación de modelos arquitecturales y mecanismos de seguridad en relación al rendimiento

Sebastián José García Fontana, Nelson Rodríguez

Departamento de Informática, Facultad de Ciencias Exactas Físicas y Naturales
Universidad Nacional de San Juan
sebasgarciafontana@gmail.com , nelson@iinfo.unsj.edu.ar

Resumen: Los sistemas de microservicios se han convertido en la actualidad en un estándar en la industria para el desarrollo, debido a las múltiples ventajas que otorgan como facilidad para el escalado, menor complejidad para implementar cambios, robustez antes caídas de servicio, entre otras. Por lo tanto no es de sorprender la popularidad de tecnologías service mesh, las cuales añaden una capa de comunicación extra entre servicios, proveyendo un mayor grado de seguridad, observabilidad y control de tráfico, abstrayendo complejidad en el networking del sistema, permitiendo a desarrolladores concentrarse en el desarrollo de aplicaciones. El presente trabajo se enfoca en los aspectos de seguridad de los service mesh, más concretamente en el impacto que genera, en latencia, consumo de CPU y de Memoria, las implementaciones de mecanismos de seguridad Mtls, Jwt y Waf en distintos modelos arquitecturales como API Gateway, Sidecar y Sidecar-Less. Los resultados indican que tanto Jwt como Mtls no presentan un impacto significativo en el rendimiento del sistema, mientras que Waf implica un gran consumo de recursos y por lo tanto debe ser utilizado con cautela.

Abstract: Microservices systems have become an industry standard for development, this because of the multiple advantages they provide, such as easier scaling, less complexity to implement changes, resilience to service failure and more. So it's of no surprise the rise in popularity of service mesh technologies, which adds an extra layer of communication between services, providing security, observability and traffic control, abstracting the complexity of the system networking, allowing developers to focus on app development. This work focuses on the security aspects of service meshes, specifically the impact in system resources, latency, CPU and Memory usage, that different security mechanisms Mtls, Jwt and Waf generate throughout multiple architectural models like API Gateway, Sidecar and Sidecar-Less. The results show that both Jwt and Mtls do not generate a significant impact in the systems performance, however Waf comes with a much higher resource consumption and therefore should be used cautiously.

Palabras clave: Microservices. Security Computing. Microservices Performance.

1. Introduction

En la actualidad existe una gran demanda para el desarrollo de sistemas de microservicios (SDM), que con el paso del tiempo se han comprobado los beneficios que presentan por encima de las arquitecturas monolíticas, por lo que no es de extrañar una expectativa de constante crecimiento de uso en la industria [1].

Los SDM funcionan creando unidades individuales llamadas servicios que trabajan de forma independiente, comúnmente llevando a cabo una sola actividad de negocio por servicio, y que luego, se comunican a través de mensajes para resolver actividades más complejas. Algunas de las ventajas que traen consigo son [2]: Independencia entre equipos desarrollando componentes separados, escalado independiente de los componentes, bajo acoplamiento el cual disminuye el impacto que puede generar la caída de un servicio en el sistema, entre otros.

Pero a pesar de sus ventajas, los SDM traen consigo otros problemas propios de su naturaleza como sistemas distribuidos, por ejemplo [2] [3] [4] [5], afirman que se produce una mayor superficie de ataque, debido a las vulnerabilidad que generan los endpoints de cada servicio, la validación de las peticiones puede impactar en el desempeño del sistema, se debe implementar confianza en los canales de comunicación entre servicios, y el trabajo necesario para el manejo de las credenciales, Surge también la necesidad de la observabilidad para poder hacer seguimiento de peticiones, etc.

Debido a la compleja naturaleza de los SDM es que existen herramientas o modelos como el API Gateway o Service Mesh. Pero toda lógica extra que se añada a un sistema implica un coste en recursos, que debido a las diversas formas en que se pueden implementar dificultan determinar cuál es el mejor caso de uso. Es por esto que en éste trabajo se ahondan en aspectos de seguridad y su impacto en la eficiencia del sistema.

2. Marco teórico

Un SDM de gran tamaño puede llegar a poseer cientos o miles de servicios en funcionamiento [6], los mismos se suelen instanciar en contenedores individuales (como una buena práctica), pero debido a su enorme cantidad, causa que se vuelve inviable la administración manual de cada instancia. Es por esto que se utilizan orquestadores, que se encargan de automatizar la administración de los contenedores, quitándole esa carga al programador, siendo Kubernetes uno de los más utilizados en la actualidad. Aun así, los servicios que brindan los orquestadores suelen ser básicos para el correcto funcionamiento y administración del sistema, debido a eso, en muchas ocasiones es deseable la implementación de otros mecanismos como por ejemplo API Gateway o Service Mesh.

Por un lado, un API Gateway facilita la comunicación entre usuario y sistema al crear un punto individual para acceder a los servicios, en lugar de necesitar múltiples puntos de acceso para cada servicio, mejorando la comunicación Norte-Sur (Usuario-Sistema). Si bien su función principal es la de enviar los mensajes a cada servicio, también es capaz de realizar otras tareas como traducción de protocolos o composición de las peticiones.

Mientras que por otro lado el modelo Service Mesh funciona como una capa extra dentro del sistema que facilita la comunicación entre los servicios dentro del sistema, mejorando la comunicación Este-Oeste (Sistema-Sistema). Como objetivo final tiene la función de facilitar la administración del tráfico en un ambiente altamente dinámico, con servicios que se crean y eliminan en tiempo de ejecución. Esto lo realizan con utilidades tales como: descubrimiento de servicios, enrutamiento y balanceo de carga interno, configuración de tráfico, encriptación, autenticación y autorización, métricas, monitoreo y observabilidad, circuit breaker o mecanismos de reintento. Todo esto provee a cerrar un sistema mucho más robusto y resistente a los constantes problemas de un sistema distribuido [2] [6].

El service mesh está compuesto por 2 planos: Un plano de información y uno de control.

El plano de información se encarga de crear el camino de datos para poder transmitir la información de las peticiones a los servicios, esto lo hace a través de proxies colocados próximos a cada servicio, a estos proxies se los suele llamar "sidecar proxy", que están encargados de llevar a cabo en tiempo de ejecución las políticas de seguridad establecidas por el plano de control.

Pero en la actualidad también está incrementando la popularidad de alternativas "sidecar-less". Dicha alternativas se integran en el service mesh directamente sobre la infraestructura en la que corren, se basan en herramientas como Daemon Sets, proxies a nivel de nodo, o tecnologías como eBPF o protocolos de aplicación a nivel de kernel. Entre sus ventajas se incluyen menor complejidad operacional o mejor eficiencia, pero como desventajas presenta falta de madurez o posibles deficiencias en seguridad comparado con un modelo sidecar [7].

Mientras que el plano de control son API y herramientas que se utilizan para manejar la configuración y el correcto funcionamiento del plano de datos, es importante no confundirlo con el plano de control del orquestador [2] [8].

Estas opciones arquitecturales para la implementaciones de comunicaciones y seguridad presentan una decisión interesante cuando se debe elegir cómo se utilizarán los mecanismos de seguridad en las distintas capas que componen el SDM, como por ejemplo el API Gateway o Service Mesh, ya sea sidecar o sidecar-less. Es por esto que en el presente trabajo se realiza un análisis de dichas implementaciones, el impacto que tienen en el rendimiento del sistema, beneficios, contras y alternativas de posibles recomendaciones.2.1. Origen y evolución

3. Trabajos relacionados

Existen estudios en los que se realizan análisis de rendimientos en SDM como:

- M. Matias et al [9], realizaron un estudio de los tiempos de respuesta cuando se utiliza SSL en un SDM.
- Y. Dawei et al. [10] plantean el uso de los algoritmos RSA y AES junto con un API Gateway para mejorar la seguridad de los sistemas, junto con un análisis del impacto en los tiempos de respuesta que generan.
- M. R. Saleh Sedghpour, C. Klein, y J. Tordsson [6] realizan la implementación de un Circuit Breaker propio en un service mesh como alternativa los Circuit breaker estáticos, el impacto en las peticiones respondidas y no respondidas, así como las diferencias en tiempos de respuesta.
- Zhu, X. et al. [11] indica como la implementación de un service mesh genera una gran cantidad de overhead al sistema que los desarrolladores muchas veces desconocen, incrementando la latencia y uso de CPU. Es por esto que realiza un estudio sobre los factores que conllevan a este overhead y desarrolla un software que ayuda a estimar el overhead generado por un service mesh.
- Y. Elkhatib and J. P. Poyato. [12] compara dos tecnologías de service mesh muy populares: Itsio y Linkerd, en el ámbito de tecnologías edge y su repercusión en los tiempos de respuesta, memoria y uso de CPU.
- A. B. Barr et al. [13] hace un análisis del consumo de recursos (Latencia, CPU y Memoria) de mTls en diferentes tecnologías service mesh de amplio uso.

En los trabajos encontrados en su mayoría realizan análisis comparativos de diferentes tecnologías service mesh sin la implementación de mecanismos de seguridad, o aquellos que si los implementan el enfoque se mantiene en una de las 2 implementaciones API Gateway o Service Mesh, ya sea sidecar o sidecar-less, sin compararlas.

Aun así es importante entender porque se elegiría una opción de implementación sobre la otra. El motivo de mayor importancia por el cual las empresas deciden implementar un service mesh es la seguridad, entre muchas otras. Pero, la performance del sistema se mantiene como un desafío técnico relevante [14], lo que lleva a las implementaciones en API Gateway, que requieren de muchos menos recursos y están enfocadas en ser mejores para la performance del sistema (para el ámbito de la seguridad).

Es aquí donde surge una problemática, si no se conoce con exactitud el consumo de cada alternativa, y los requisitos de seguridad del sistema, se puede dar una situación donde la implementación de un service mesh resulte excesivo produciendo un overhead que dañe la experiencia del usuario, genere incrementos en los costos operacionales y decrementos en los ingresos [11]. O por el contrario que una implementación solo de gateway, enfocado solo en la búsqueda de maximizar performance provoque falencias en la seguridad del sistema, y por extensión del usuario.

Es por esto que un análisis cuantitativo puede resultar útil para la ayuda en la toma de decisiones, facilitando así poder determinar si la elección de una implementación se ajusta a los objetivos que se esperan del sistema.

4. Metodología y desarrollo

Se seleccionó una metodología experimental. Desarrollando un sistema que intenta simular lo máximo posible un cluster pequeño real de producción de kubernetes, sobre el cual aislar cada uno de los mecanismos a evaluar y realizar pruebas. Dichas pruebas consisten en generar peticiones http/s que lleven al límite al sistema para poder obtener métricas cuantitativas sobre el consumo de recursos, con las que luego comparar las distintas implementaciones.

Para el desarrollo se eligió kind para la implementación de k8s, por su facilidad de uso y haber sido desarrollado principalmente para el testing de k8s. Dicha implementación consiste en un cluster con 4 nodos, 1 maestro sobre el que corren los servicios propios de K8s y 3 worker sobre el que corren las aplicaciones desarrolladas para las pruebas.

Para la implementación del API Gateway y Service mesh se optó por Istio debido a su facilidad de uso, capacidad para implementar ambos modelos arquitecturales (sidecar y sidecar-less) y soporte nativo para los mecanismo de seguridad a evaluar (Jwt, mTls, y Waf). Istio utiliza proxies de Envoy para desplegar los sidecar de cada pod en su modo sidecar, mientras que utiliza uno por nodo en su modo ambiente.

Se desarrollaron 3 aplicaciones que se desplegaron sobre el sistema. Primero una aplicación de “Cpu-bench” que busca poner en estrés la cpu realizando una operación de multiplicación de matrices de órdenes elevadas. Segundo una aplicación “Mem-bench” que pone en estrés el uso de memoria al correr un script en C que aloje un bloque de gran tamaño en memoria para luego liberarlo. Y finalmente una aplicación “Front” que sea las que reciba las peticiones y las redirige a los otros 2 servicios. Se optó por desplegar 3 copias de cada servicio.

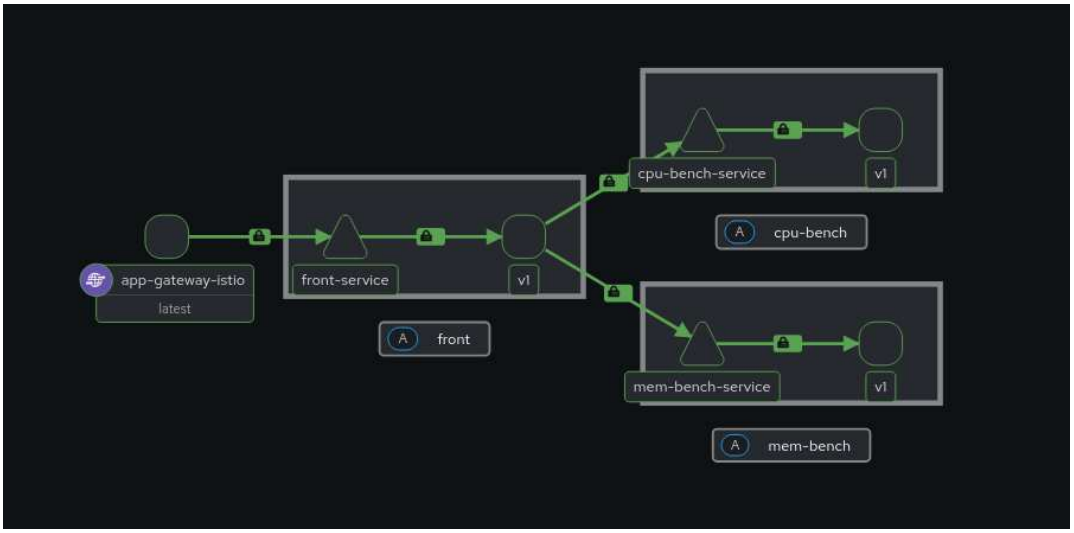


Figura 1: Flujo de ejecución del sistema desarrollado, tomado desde servicio de monitoreo Kiali

Una vez desplegado el sistema se utilizó autocannon como generador de peticiones http/s, con una carga de 10000 peticiones, 100 concurrentes. Después de cada prueba se utilizó el servicio de monitoreo Prometheus para obtener las métricas de consumo de CPU y Memoria del sistema. Se corrió 20 pruebas para cada mecanismo en cada una de sus implementaciones (Gateway, Sidecar y Sidecar-less).

Aclaración: Cuando se hace referencia a Mtls en los sidecar implica comunicación mtls entre todos los pods, incluido las comunicaciones del gateway hacia el lado del sistema. La implementaciones de Mtls en el gateway se debieron implementar como tls simple, a través de https, para las comunicaciones hacia el lado del cliente.

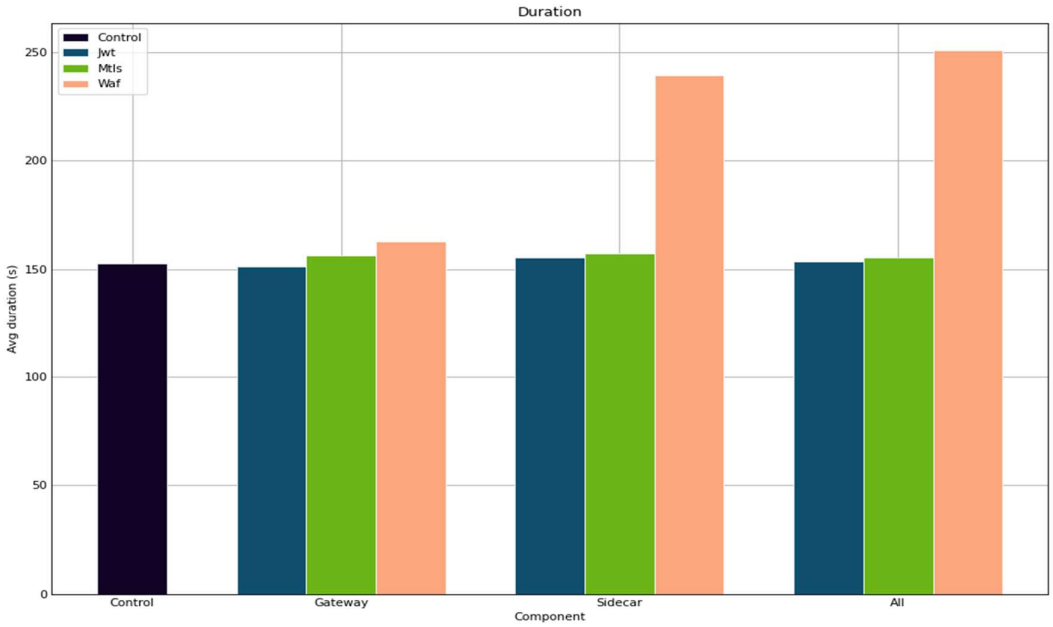
Todo el código fuente del trabajo se puede encontrar en:
https://github.com/Sakarrex/Microservices-security-evaluation

Especificaciones:

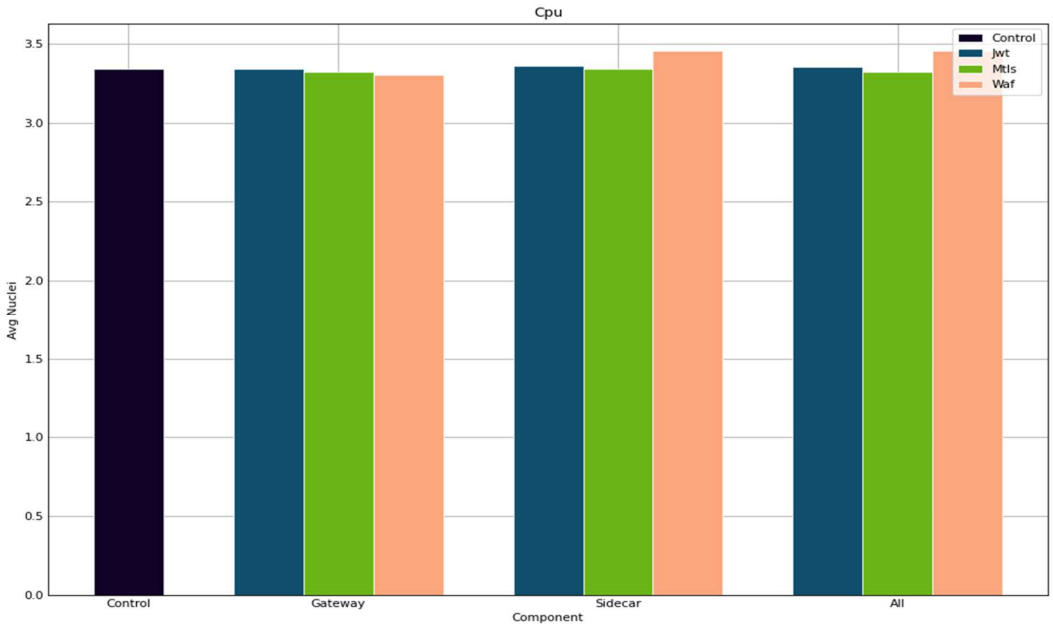
- Máquina virtual: Ubuntu 24.04.3
- Núcleos: 4
- Memoria: 13 GB
- Versión de kubernetes: 1.35.3
- Versión de istio: 1.29.1

5. Resultados

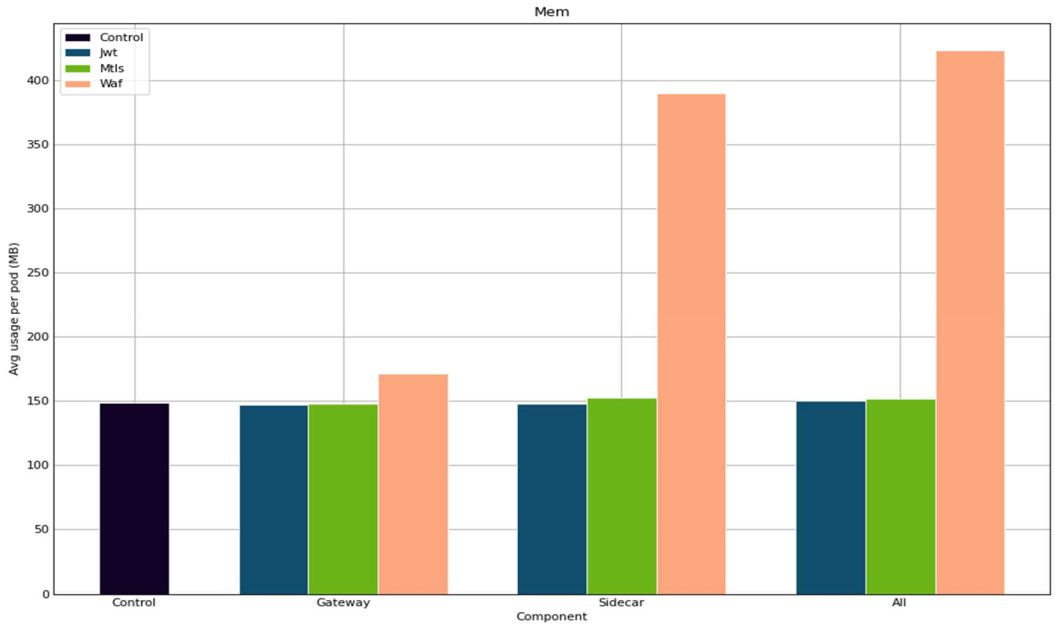
Istio modo sidecar



(a) Duración



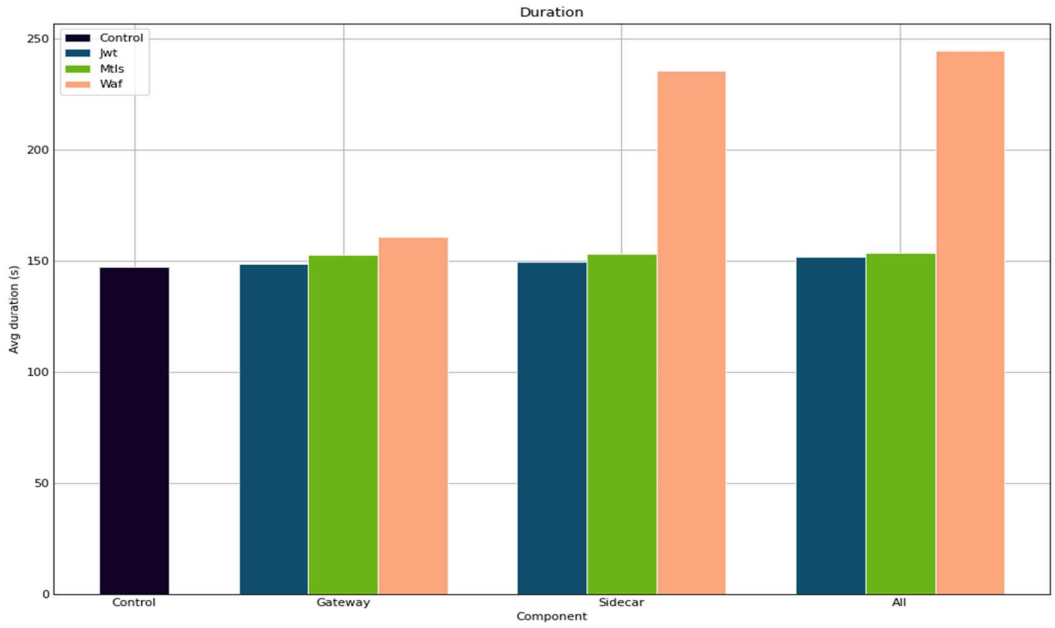
(b) Uso de CPU



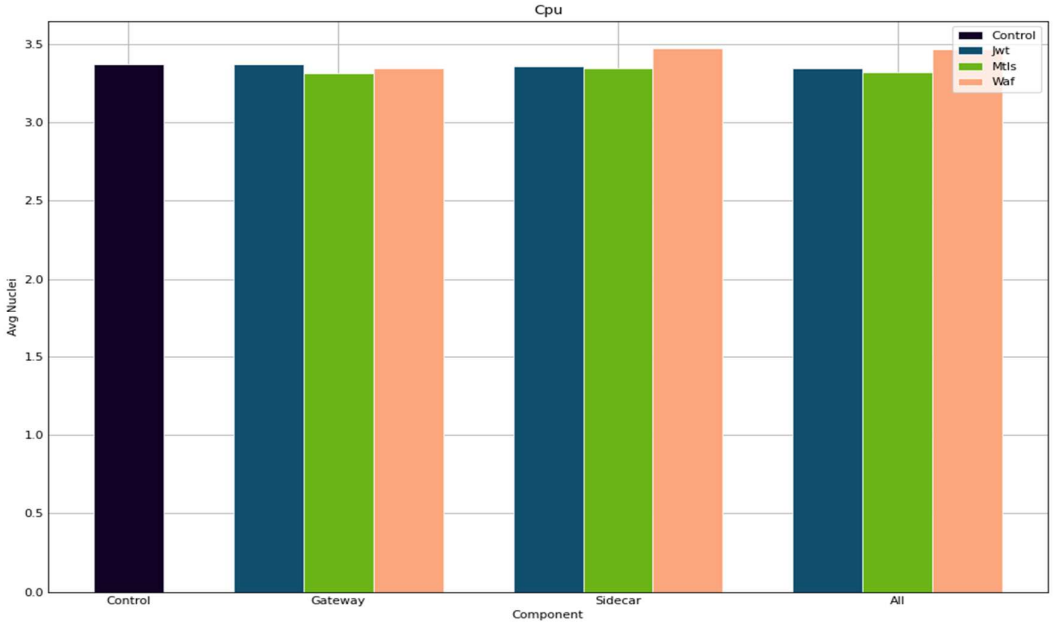
(c) Memoria

Figura 2: Resultados Istio modo sidecar

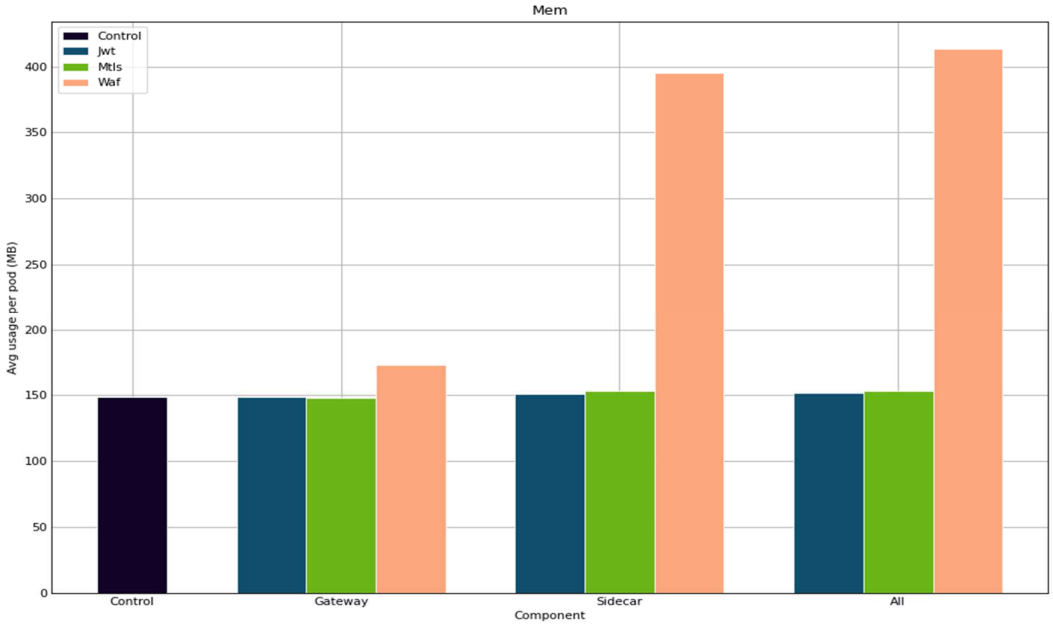
Istio modo ambiente



(a) Duración



(b) CPU



(c) Memoria

Figura 3: Resultados Istio modo ambiente

De los resultados se puede observar lo siguiente:

- Todas las implementaciones de Jwt y Mtls tienen consumos similares tanto en su duración, de Cpu y de Memoria. A su vez este consumo se aproxima al de las pruebas de control.
- Las implementaciones de Waf son muchas más caras para todos los suponiendo un aumento aproximadamente un 66% en la duración y 166% en la memoria, aunque ligeramente menor en cpu de un 3%.
- Las implementaciones sidecar less son ligeramente más eficientes en consumo, pero se mantiene la proporción del consumo de recursos para cada implementación.

6. Conclusiones

En conclusión, según los resultados obtenidos, el único mecanismo que realmente tiene un impacto en el rendimiento del sistema es Waf. Esto debido a que para implementar una solución Waf implica añadir a un pod toda la lógica extra de un firewall con todas sus reglas, lo cual si se implementan sólo en el gateway no implica gran diferencia, pero si se deben implementar en cada uno de los pods que corren dentro sistema supone un costo de recursos importante.

Por lo tanto mecanismos de Jwt y Mtls deberían ser implementadas siempre que resulte posible, mientras que Waf solo debería aplicarse al gateway a unos pocos servicios que sea de vital importancia el mantener protegidos.

Amenazas a la validez:

Durante la decisión de implementar un service mesh un desarrollador encuentra muchas alternativas que decidir. Desde las más grandes como el servicio proveedor (Istio, Ciluim, Linkerd, etc) hasta configuraciones específicas del comportamiento de cada implementación, lo cual vuelve muy complejo poder medir el comportamiento del sistema en cada una de las implementaciones posibles. Inclusive considerando como cada sistema tiene sus propios requisitos una misma configuración que resulte eficiente en uno, genere problemas en el rendimiento de otro. Es por esto que realizar pruebas en cada sistema y tener un monitoreo robusto de su funcionamiento es de gran importancia

Trabajo a futuro:

- Realizar pruebas sobre sistemas de mayor escala.
- Comprobar si los resultados se mantienen con otras tecnologías.
- Autenticación que se adapte a tiempo real.

Referencias

[1] "Cloud Microservices Market Size, Share | 2022–27 | Industry Analysis," *Mordor Intelligence*. [Online]. Available: https://www.mordorintelligence.com/industry-reports/cloud-microservices-market
[2] R. Chandramouli, "Security Strategies for Microservices-Based Application Systems," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST SP 800-204, Aug. 2019, doi: https://doi.org/10.6028/nist.sp.800-204.

[3] M. Souppaya, J. Morello, and K. Scarfone, "Application Container Security Guide," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST SP 800-190, Sep. 2017, doi: https://doi.org/10.6028/nist.sp.800-190.
[4] P. Siriwardena and N. Dias, Microservices Security in Action. Shelter Island, NY: Manning Publications, 2020.
[5] M. G. de Almeida and E. D. Canedo, "Authentication and Authorization in Microservices Architecture: A Systematic Literature Review," Appl. Sci., vol. 12, no. 6, p. 3023, Mar. 2022, doi: https://doi.org/10.3390/app12063023.
[6] M. R. Saleh Sedghpour, C. Klein, and J. Tordsson, "An Empirical Study of Service Mesh Traffic Management Policies for Microservices," Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering, Apr. 2022, doi: https://doi.org/10.1145/3489525.3511686.
[7] I. McPhee, "Sidecarless Service Meshes – Are They Ready for Prime Time?" GigaOm, 2026. [Online]. Available: https://portal.gigaom.com/blog/sidecarless-service-meshes-are-they-ready-for-prime-time (accessed Apr. 17, 2026).
[8] R. Chandramouli and Z. Butcher, "Building Secure Microservices-Based Applications Using Service-Mesh Architecture," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST SP 800-204A, May 2020, doi: https://doi.org/10.6028/nist.sp.800-204a.
[9] M. Matias, E. Ferreira, N. Mateus-Coelho, O. Ribeiro, and L. Ferreira, "Evaluating Effectiveness and Security in Microservices Architecture," Procedia Computer Science, vol. 237, pp. 626–636, 2024, doi: https://doi.org/10.1016/j.procs.2024.05.148.
[10] Y. Dawei, G. Yang, H. Wei, and L. Kai, "Design and Achievement of Security Mechanism of API Gateway Platform Based on Microservice Architecture," Journal of Physics: Conference Series, vol. 1738, p. 012046, Jan. 2021, doi: https://doi.org/10.1088/1742-6596/1738/1/012046.
[11] X. Zhu et al., "Dissecting Overheads of Service Mesh Sidecars," in Proc. ACM Symposium on Cloud Computing (SoCC '23), Santa Cruz, CA, USA, Oct. 2023, pp. 142–157, doi: https://doi.org/10.1145/3620678.3624652.
[12] Y. Elkhatib and J. P. Poyato, "An Evaluation of Service Mesh Frameworks for Edge Systems," in Proc. ACM/IFIP International Middleware Conference (Middleware '23), Bologna, Italy, Apr. 2023, doi: https://doi.org/10.1145/3578354.3592867.
[13] A. B. Barr, O. Lavi, Y. Naor, S. Rampal, and J. Tavori, "Technical Report: Performance Comparison of Service Mesh Frameworks: the MTLS Test Case," arXiv, preprint arXiv:2411.02267, 2024. [Online]. Available: https://arxiv.org/abs/2411.02267
[14] Cloud Native Computing Foundation (CNCF), "Service Meshes Are on the Rise – But Greater Understanding and Experience Are Required," CNCF MicroSurvey, 2022. [Online]. Available: https://www.cncf.io/wp-content/uploads/2022/05/CNCF_Service_Mesh_MicroSurvey_Final.pdf