Perbandingan Kinerja *Service Mesh* Pada Manajemen *Microservices* di Kubernetes

Muhamad Arif Rahman Hakim[1], I Made Suartana[2]

[1,2] Program Studi Teknik Informatika/Teknik Informatika, Universitas Negeri Surabaya
[1]muhamad.19085@mhs.unesa.ac.id
[2]madesuartana@unesa.ac.id

***Abstrak*— Seiringnya pertumbuhan bisnis dan kebutuhan teknologi yang kian meningkat, menciptakan tantangan tersendiri bagi dunia bisnis dan digital. Berbagai macam arsitektur dan metode perancangan sistem aplikasi bermunculan hingga yang saat ini sedang populer adalah penggunaan arsitektur *microservices* yang memungkinkan pengembangan aplikasi yang lebih fleksibel dan mudah dikelola, dengan menjadikan setiap fitur aplikasi sebagai entitas terpisah yang dapat dikembangkan dan di-*deploy* secara independen. Pengelolaan aplikasi *microservices* tentu membutuhkan *tools* yang mudah untuk digunakan, di *maintenance*, dan dapat di kelola secara otomatis sehingga banyak bermunculan *tools* manajemen *microservices* salah satunya adalah kubernetes. Namun, kubernetes dinilai masih mempunyai banyak kekurangan dan minim fitur, sebagai contoh pada pengelolaan komunikasi antar *microservices. Service mesh* muncul sebagai salah satu solusi pengelolaan komunikasi antar *microservices* yang dapat mengatasi masalah komunikasi antar aplikasi mikro. Penelitian ini bertujuan membandingkan performa *microservices* yang menggunakan *service mesh* tambahan dan *service mesh* bawaan kubernetes. Pengujian akan dilakukan pada lingkungan *cloud* yang memiliki ketahanan lebih baik dibanding lingkungan lokal. Berdasarkan hasil pengujian yang dilakukan ternyata penggunaan *service mesh* untuk pengelolaan komunikasi antar *microservices* memiliki performa lebih baik, namun dengan penggunaan sumber daya yang sedikit lebih besar dikarenakan terdapat komponen tambahan pada *service mesh* Istio atau *service mesh* tambahan.**

***Kata Kunci* - *Service mesh*, Kubernetes, Istio, *Microservices*, performa, Arsitektur.**

I. PENDAHULUAN

Pertumbuhan *e-commerce* dan bisnis di bidang digital yang sangat cepat menciptakan tantangan tersendiri bagi dunia digital dan berbagai aspek bisnis lainnya untuk beradaptasi dan menyesuaikan diri dengan perkembangan tersebut. Ini sangat penting untuk memastikan bahwa perusahaan dapat terus berkembang dan memenuhi kebutuhan pelanggan mereka.

Aplikasi berbasis *microservices* hadir dan menjadi salah satu solusi yang digunakan oleh banyak perusahaan demi memenuhi kebutuhan konsumen. *Microservices* memungkinkan pengembangan aplikasi yang lebih fleksibel dan mudah dikelola, dengan menjadikan setiap fitur aplikasi sebagai entitas terpisah yang dapat dikembangkan dan di-*deploy* secara independen [1], Dalam beberapa tahun terakhir terjadi peningkatan yang signifikan dalam tren arsitektur pengembangan menggunakan microservice, seperti yang ditunjukkan oleh hasil Google Trends sejak tahun 2018.

Pengelolaan *microservices* tentu memerlukan tools yang dapat menjalankan tugas-tugas seperti *deployment, scaling, monitoring,* dan *maintenance* aplikasi dengan mudah dan otomatis untuk para pengembang. Platform *container orchestration* atau orkestrasi kontainer adalah platform yang memungkinkan manajemen dan pengelolaan kontainer secara otomatis dan mudah, termasuk melakukan tugas-tugas seperti *deployment, scaling, monitoring,* dan *maintenance* aplikasi [2]. Platform ini memungkinkan pengembang dan administrator sistem untuk memfokuskan upaya industri atau perusahaan pada pengembangan aplikasi dan meningkatkan kualitas layanan yang diberikan kepada pengguna, sementara tugas-tugas operasional seperti manajemen dan pengelolaan infrastruktur ditangani oleh platform orkestrasi kontainer. Beberapa contoh platform *container orchestration* yang populer antara lain Kubernetes dan Docker Swarm [3]. Namun, dengan seiring berjalannya waktu dan kompleksitas deployment tentu akan menyebabkan masalah dalam manajemen komunikasi antar kontainer, tracing (pelacakan) jika ada masalah pada salah satu dari ratusan kontainer yang beroperasi, pengamatan pada kontainer, keamanan, toleransi kesalahan, *routing*, dan konsistensi saat komunikasi antar kontainer atau *services* [4].

Untuk mengatasi permasalahan komunikasi pada *microservices* terdapat beberapa solusi yang dapat digunakan, salah satunya adalah dengan menggunakan metode *service mesh* seperti Istio atau Linkerd. *Service mesh* tambahan seperti Istio memiliki berbagai fitur diantaranya: *routing, observability, tracing, dan security built-in,* dan masih banyak fitur yang ditawarkan. *Service Mesh* mempermudah pengelolaan *microservices* di Kubernetes dengan memfasilitasi komunikasi antar *microservices* sehingga dapat menjadi solusi untuk masalah komunikasi dalam klaster kubernetes [5].

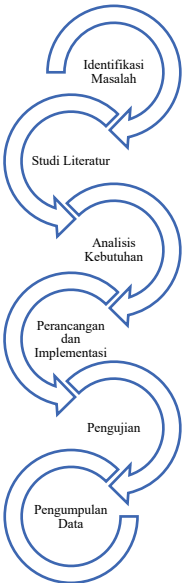
Penelitian ini akan membandingkan kinerja *service mesh* pada *microservices* di kubernetes, dengan menganalisis faktor-faktor seperti *latency, throughput,* dan *resource utilization*. Hasil analisis ini diharapkan dapat memberikan *insight* atau gambaran tentang manfaat dan kelebihan menggunakan *service mesh* dalam pengelolaan *microservices* di Kubernetes.

II. METODOLOGI PENELITIAN

A. Metode penelitian

Metode penelitian yang digunakan dalam perbandingan kinerja *microservices* di kubernetes menggunakan *service mesh* bawaan dan *service mesh* tambahan merupakan metode penelitian eksperimen yang dilakukan di lingkungan yang terkontrol, yaitu pada sebuah cluster Kubernetes yang telah

diatur dengan konfigurasi dan aplikasi yang sama pada kedua jenis *service mesh*.

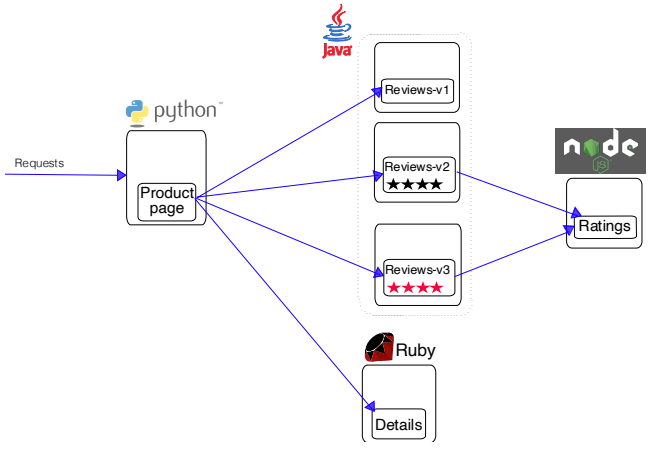


Gbr. 1 Alur Metode Penelitian

Eksperimen akan melibatkan beban kerja terdistribusi dengan jenis *front end* dan *backend* dan akan dilakukan dengan memvariasikan parameter tertentu seperti jumlah kontainer, skala aplikasi, atau jenis protokol. Juga parameter yang akan di coba pada *service mesh* bawaan maupun *service mesh* tambahan seperti *response time, error rate,* manajemen lalu lintas, *throughput,* dan *resource utilization.*

B. *Perancangan Sistem*

Aplikasi *microservices* yang digunakan merupakan contoh aplikasi yang telah dimodifikasi berdasarkan jenis *service mesh* yang akan digunakan. Layanan mikro yang menjadi contoh adalah *bookinfo app official* dari Istio dengan skema sebagai berikut :

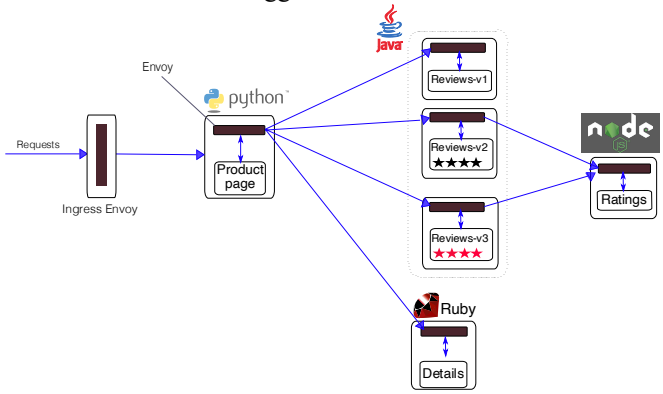


Gbr. 2 Rancangan aplikasi mikro dengan *sample app bookinfo* menggunakan *service mesh* bawaan

Pada Gbr. 2 merupakan rancangan *microservices* yang akan di *deploy* pada lingkungan tanpa *service mesh* tambahan, *microservices bookinfo* terdiri dari empat bagian atau yang biasa disebut *service* yaitu :

1) *Product Page*
2) *Details*
3) *Reviews*
4) *Rating*

Mekanisme dari skema rancangan pada Gbr. 2 yaitu ketika *user request* atau melakukan permintaan ke dalam halaman utama atau beranda maka *service productpage* akan melakukan permintaan atau *request* ke masing-masing *service* yang berhubungan, sehingga *service* lainnya akan mengembalikan data yang kemudian ditampikan di halaman utama *productpage*. Semua aktivitas pada mekanisme komunikasi diatas menggunakan *service mesh* bawaan kubernetes, lain halnya dengan mekanisme dan skema rancangan yang berada pada *service mesh* tambahan menggunakan Istio. Yaitu :



Gbr. 3 Rancangan aplikasi mikro dengan *sample app bookinfo* menggunakan *service mesh* tambahan

(Sumber:https://istio.io/latest/docs/examples/bookinfo/)

Pada Gbr. 3 kita bisa melihat bahwa pada masing-masing *service* terdapat *sidecar* atau kontainer tambahan yang melekat pada masing-masing *service* yang disebut *envoy proxy*.

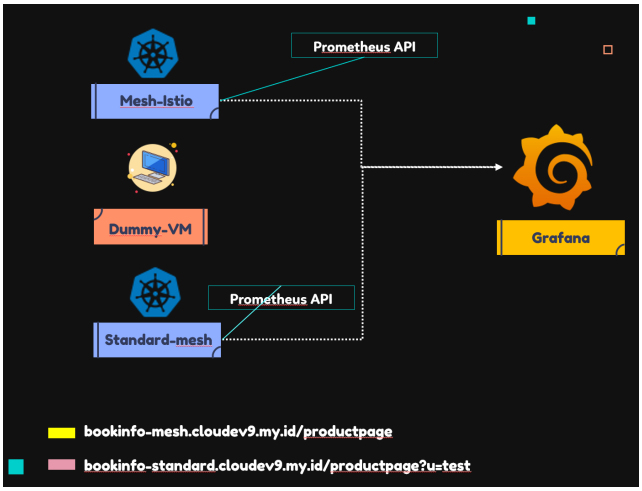
Envoy proxy ini bertugas sebagai *agent* komunikasi antar *service* yang juga memiliki kemampuan tambahan atau fitur tambahan berdasarkan komponen Istio *service mesh*. Dengan Istio, pengembang dapat mengatur lalu lintas layanan, keamanan, dan mengamati layanan secara lebih teratur dan terstruktur dalam jaringan [7].

Mekanisme dari *microservices* yang menggunakan *service mesh* tambahan adalah dengan setiap *service* berkomunikasi secara tidak langsung dengan *service* lain, melainkan komunikasi akan di ambil alih oleh *envoy proxy* kemudian *envoy proxy* akan memproses permintaan dan konfigurasi yang sudah didefinisikan, konfigurasi yang dimaksud adalah pendefinisian objek objek *custom* yang di install bersama *service mesh* tambahan Istio. Seperti *Ingress Gateway* yang berfungsi sebagai jalan masuk pertama untuk *request* yang masuk, *Virtual Service* yaitu memungkinkan perutean komunikasi setelah request dari user melewati gateway. Objek-objek konfigurasi ini lalu kemudian diteruskan menuju *envoy proxy service* yang dituju. Sehingga komunikasi akan menjadi lebih cepat karena memiliki *tunnel* khusus.

C. Skenario Pengujian

Pada skenario yang akan digunakan sebagai pengujian yaitu pembuatan dummy-VM yang satu region dengan GKE *(Google Kubernetes Engine)* berada. Misalnya GKE berada di *region asia-southeast1-c* maka dummy-VM juga harus berada pada *region* tersebut. Ini dimaksudkan agar hasil pengujian yang dilakukan memiliki hasil yang optimal.

Dummy-VM berisi *script* dari k6-test yang ditulis menggunakan bahasa javascript dan dapat dijalankan dengan mode *cloud* maupun lokal. *Cloud* disini adalah layanan premium dari k6-test yang memungkinkan hasil dari pengujian dapat langsung di analisis dan dibaca dengan mudah, namun dengan biaya layanan yang mengharuskan kita berlangganan. Sedangkan jika kita menjalankan pengujian di mode lokal kita harus mengkonfigurasi *output,, metrics,* dan memvisualisasikan *metrics* pengujian kita sendiri *(self managed)*.



Gbr. 4 Ilustrasi Skenario Pengujian

Pada Gbr. 4 merupakan ilustrasi skenario pengujian, dapat di lihat bahwa dummy-VM akan menjalankan script k6-test, yang kemudian pada setiap klaster kubernetes memiliki *scaper* yaitu prometheus sebagai pengumpul *metrics* dan dikirimkan ke Grafana untuk kemudian di visualisasi.

Script yang ditulis berisi *endpoint* atau alamat IP eksternal dari *microservices* yang sudah di *deploy* ke masing-masing klaster sesuai parameter yang akan di uji. Terdapat empat parameter yang akan di uji yaitu *response time, error rate, throughput,* dan pemakaian sumber daya.

III. Hasil dan Pembahasan

Pada tahap ini akan disajikan beberapa hasil dan juga skenario yang digunakan untuk mengetahui performa antara kedua *service mesh* yang sudah di bangun dengan jumlah *user request* 300 sampai dengan 500 *user request*.

A. Traffic 300 User Request

Pada hasil pengujian dengan 300 user request bisa kita lihat pada TABEL I, dengan parameter-parameter yang sudah di tentukan di awal skenario pengujian.

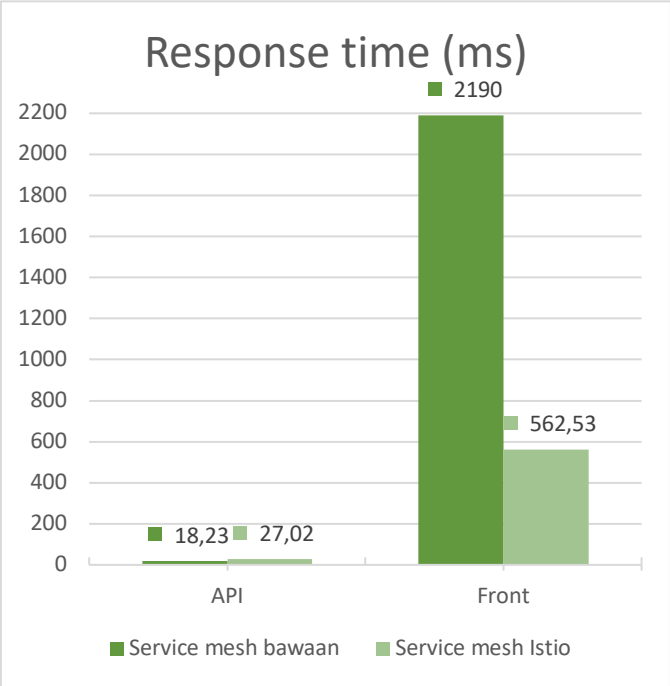
TABEL I
HASIL PERBANDINGAN PENGUJIAN 300 *USER*

Service mesh	Endpoint	Response time (ms)	Error Rate (%)	Through put (req/s)
Bawaan	API	18,23 ms	0	214,63
	front page	2.190 ms	0,031	68,42
Istio	API	27,02 ms	0	213,42
	front page	526,53 ms	0,08	25,42

Parameter *response time* merupakan waktu total yang dibutuhkan *packet* untuk menanggapi permintaan layanan, termasuk waktu layanan dan waktu tunggu. Biasanya satuan dari *response time* adalah dalam mili detik (ms)

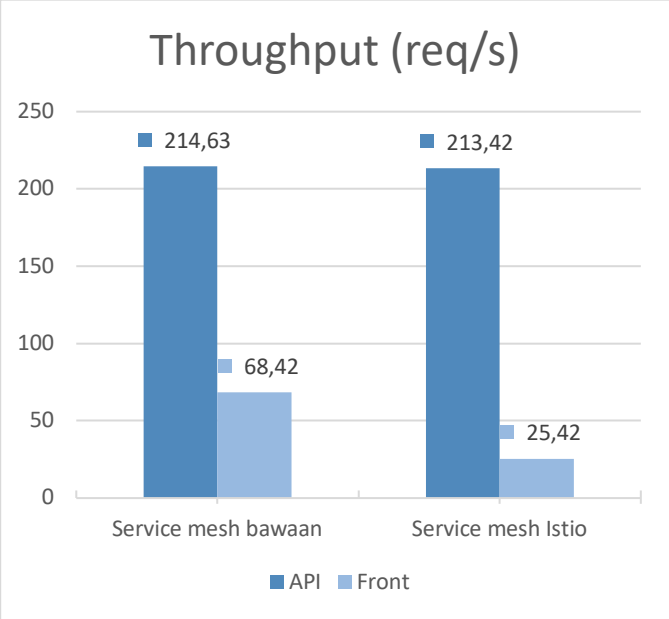
Parameter *error rate* berfungsi untuk mengamati berapa persentase kegagalan permintaan dalam keberlangsungan pengujian beban ke sistem.

Throughput sendiri adalah seberapa cepat permintaan per detik yang terkirim dengan persamaan rumus *Throughput* = Jumlah total permintaan / Waktu total pengujian. Biasanya satuan dari *throughput* sendiri bisa berupa Kbps, Gbps, atau juga bisa dalam req/s.



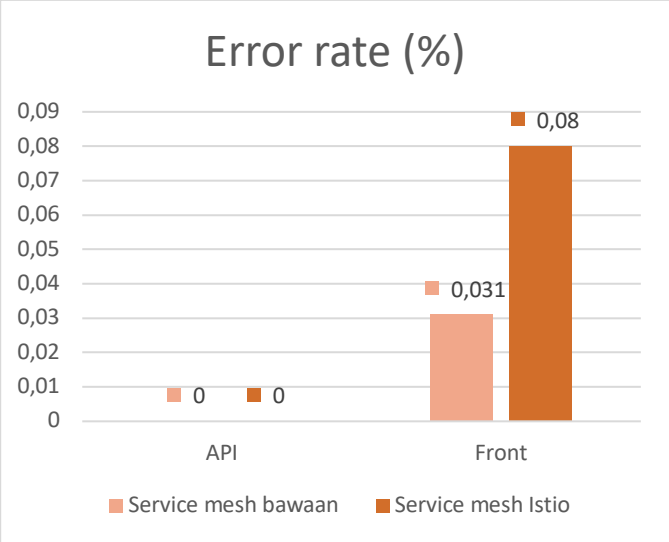
Gbr. 5 Perbandingan hasil *response time* 300 user

Pada hasil pengujian dengan 300 VUs dalam TABEL I dan grafik pada Gbr. 5, dapat diamati bahwa pada parameter *response time* untuk *endpoint API* antara *service mesh* bawaan dan Istio terjadi selisih 8,79 ms (mili detik). Sedangkan pada *endpoint front page* atau halaman utama mendapat selisih *response time* 1663,47ms atau sekitar 1,6 detik lebih cepat pada *service mesh* yang menggunakan Istio.



Gbr. 6 Perbandingan hasil *throughput* 300 user

Kemudian bisa kita amati di Gbr. 6, pada endpoint API, parameter throughput terjadi selisih tidak terlalu signifikan hanya 1,21 req/s. Sedangkan pada endpoint front page untuk throughput mendapat selisih 43 req/s lebih besar pada service mesh.

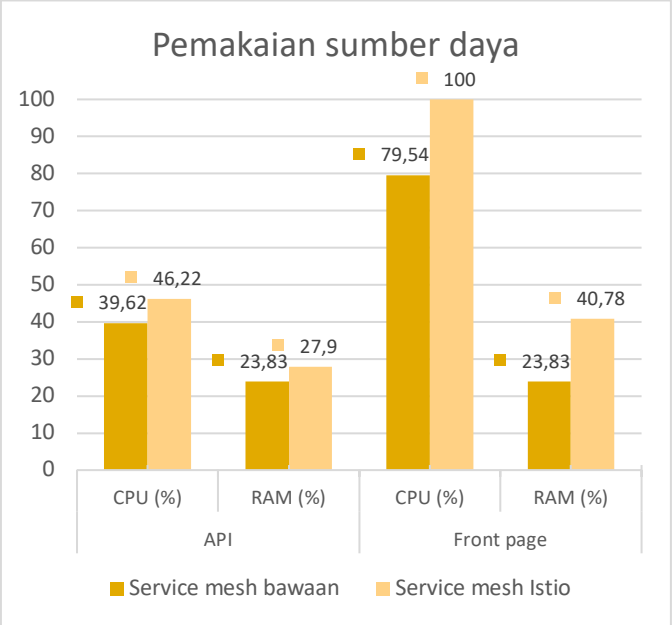


Gbr. 7 Perbandingan hasil *error rate* 300 user

Untuk hasil perbandingan parameter *error rate* di Gbr. 7, pada bagian *endpoint API* tidak mengalami *error* sama sekali yaitu 0%, hal tersebut juga terjadi untuk *endpoint front page* yang hampir tidak terdapat *error* sama sekali hanya selisih tidak sampai 0,1 %.

Tabel II
HASIL PERBANDINGAN PENGGUNAAN SUMBER DAYA 300 USER

Service mesh	Endpoint	CPU (%)	RAM (%)
Bawaan	API	39,62	23,83
	front page	79,54	23,83
Istio	API	46,22	27,9
	front page	100	40,78



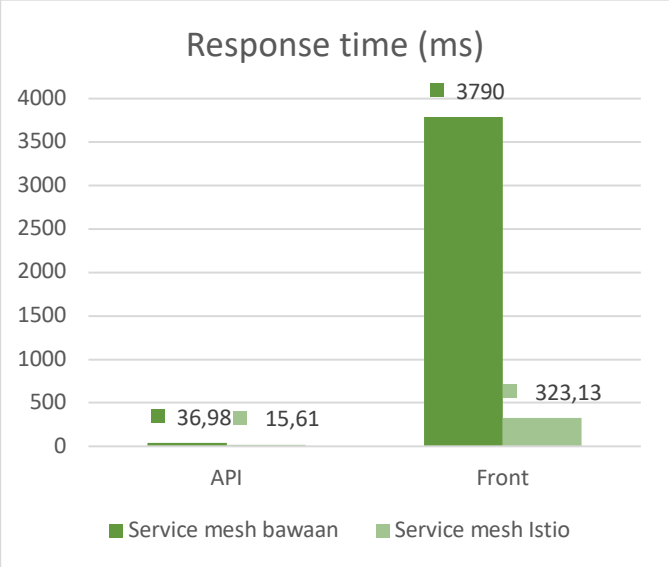
Gbr. 8 Perbandingan hasil pemakaian sumber daya 300 user

Pada parameter pamakaian sumber daya TABEL II dan Gbr. 8 grafik perbandingan, terlihat bahwa *service mesh* yang menggunakan Istio menggunakan sumber daya yang sedikit lebih banyak ditunjukkan pada *endpoint API* menggunakan 6,6% CPU dan 4,07% RAM lebih banyak dari *service mesh* bawaan. Pada *endpoint front page* menggunakan 20,46% CPU dan 16,95% RAM lebih besar dari pada *service mesh* bawaan.

B. Traffic 500 User Request

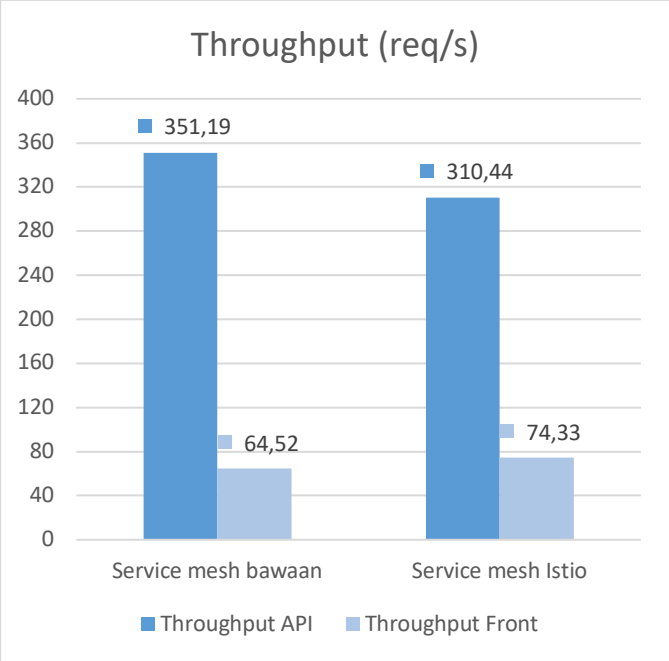
Tabel III
HASIL PERBANDINGAN PENGUJIAN 500 USER

Service mesh	Endpoint	Response time (ms)	Error Rate (%)	Throughput (req/s)
Bawaan	API	36,98 ms	0	351,19
	front page	3.790 ms	8,21	64,52
Istio	API	15,61 ms	0	310,44
	front page	323,13 ms	0	74,33



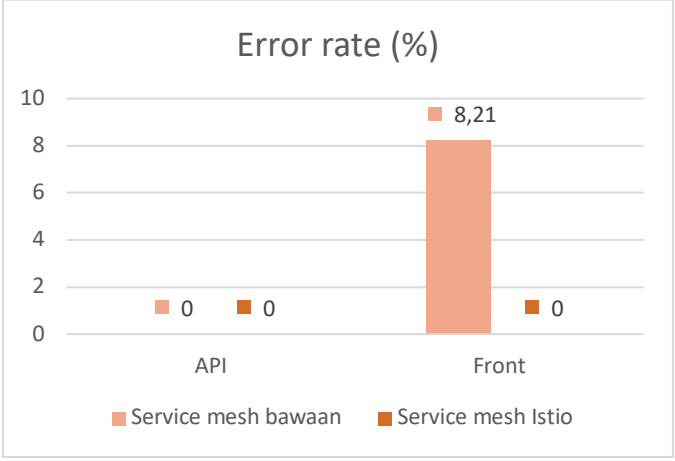
Gbr. 9 Perbandingan hasil response time 500 user

Pada hasil pengujian dengan 500 VUs dalam TABEL III dan Gbr. 9 grafik perbandingan, dapat diamati bahwa pada parameter *response time* untuk *endpoint API* antara *service mesh* bawaan dan Istio terjadi selisih 21,37 ms (mili detik). Sedangkan pada *endpoint front page* atau halaman utama mendapat selisih *response time* cukup besar sekitar 3466,87 ms atau sekitar 3,5 detik lebih cepat pada *service mesh* yang menggunakan Istio.



Gbr. 10 Perbandingan hasil *throughput* 500 user

Kemudian pada endpoint API, parameter *throughput* Gbr. 10, terjadi selisih 40,75 req/s lebih besar di *service mesh* bawaan. Sedangkan pada *endpoint front page* untuk *throughput* mendapat selisih sekitar 10 req/s lebih besar pada s*ervice mesh* Istio.

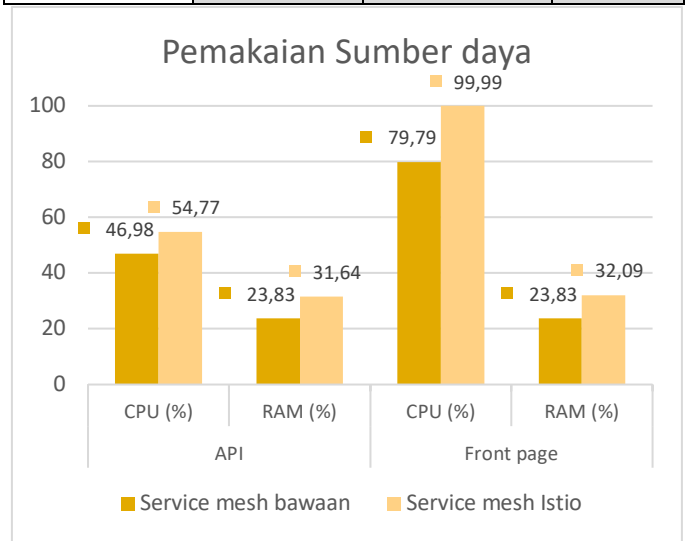


Gbr. 11 Perbandingan hasil *error rate* 500 user

Untuk hasil perbandingan parameter *error rate* pada bagian *endpoint API* tidak mengalami *error* sama sekali yaitu 0%, hal tersebut menjadi cukup menarik untuk *endpoint front page* karena terdapat selisih *error rate sebesar* 8,21% yang terjadi di *service mesh* bawaan.

Tabel IV
HASIL PERBANDINGAN PENGGUNAAN SUMBER DAYA 500 USER

Service mesh	Endpoint	CPU (%)	RAM (%)
Bawaan	API	46,98	23,83
	front page	79,79	23,83
Istio	API	54,77	31,64
	front page	99,99	32,09



Gbr. 12 Perbandingan hasil pemakaian sumber daya 500 user

Pada parameter pamakaian sumber daya TABEL IV dan Gbr. 12 grafik perbandingan, terlihat bahwa *service mesh* yang menggunakan Istio menggunakan sumber daya yang sedikit lebih banyak ditunjukkan pada *endpoint* API menggunakan 7,79% CPU dan 7,81% RAM lebih banyak dari *service mesh*

bawaan. Pada *endpoint front page* menggunaan 20,2 % CPU dan 8,26% RAM lebih besar dari pada *service mesh* bawaan.

C. Perbandingan Fitur Tambahan

Penggunaan *service mesh* tambahan selain mempertimbangkan performa dan efisiensi, fitur yang dimiliki oleh *service mesh* tambahan juga menjadi pertimbangan yang sangat penting. Istio memungkinkan pengelolaan jaringan tingkat lanjut karena memiliki beberapa fitur yang dapat dikategorikan menjadi tiga bagian, yaitu pengamatan *(observability)*, manajemen lalu lintas *(traffic management)*, dan keamanan *(security)* [6]. Berikut adalah perbandingan dan pertimbangan fitur yang di miliki oleh *service mesh* tambahan Istio dengan bawaan kubernetes.

Tabel V
PERBANDINGAN FITUR *SERVICE MESH* BAWAAN DAN ISTIO *SERVICE MESH*

No.	Fitur	Istio Service Mesh	Service Mesh Bawaan
1	*Traffic Management*	• Request Routing • Fault Injection • Traffic Shifting • Request Timeout • Circuit Breaking • Ingress • Egress • Mirroring	• Service • Ingress
2	*Security*	• Certificate Management • Authentication • Authorization • TLS Configuration	• None
3	*Observability*	• Metrics • Logs • Distributed Tracing • Visualization	• None

IV. KESIMPULAN

Mengacu pada perancangan sistem dan skenario pengujian kinerja *service mesh* dalam manajemen *microservice* di Kubernetes, terlihat bahwa layanan mikro yang menggunakan Istio sebagai *service mesh* memiliki kinerja lebih baik yang ditunjukkan pada pengujian 500 *user request* untuk parameter *error rate*, meskipun hasil perbandingan 300 *user request* juga menunjukkan hasil yang sedikit menunjukkan *service mesh* bawaan lebih baik di parameter *response time*. Hal ini ditunjukkan oleh parameter-parameter lainnya yang didapatkan, baik pada *endpoint* API maupun *front page*. Namun, performa yang lebih baik pada *microservices* yang menggunakan *service mesh* Istio sejalan dengan penggunaan sumber daya klaster kubernetes. Bahkan bisa jadi konsumsi untuk VM yang disediakan akan bertambah dikarenakan penggunaan yang intens dan kompleks dengan ditunjukkan pada hasil pengujian yang sudah dilakukan. Pemilihan *service mesh* Istio juga di dukung dengan berbagai fitur yang ditawarkan karena dengan fitur yang di tawarkan oleh Istio dapat menjadikan produktivitas *developer* atau pelaku usaha yang menggunakan arsitektur *microservice* dapat meningkat.

REFERENSI

[1] Paolo Di Francesco, “Architecting Microservices,” Apr. 2017, doi: https://doi.org/10.1109/icsaw.2017.65.
[2] R. W. K, “Implementasi Mutual Transport Layer Security (mTLS) Pada Arsitektur Microservices Dengan Istio Di Kubernetes,” *dspace.uii.ac.id*, 2020, Accessed: Jul. 03, 2023. [Online]. Available: https://dspace.uii.ac.id/handle/123456789/28351
[3] Raouf Boutaba, “The cloud to things continuum,” Dec. 2021, doi: https://doi.org/10.1145/3501255.3501407.
[4] R. Mara Jösch, *Managing Microservices with a Service Mesh : An implementation of a service mesh with Kubernetes and Istio*. 2020. Accessed: Jul. 03, 2023. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-280407
[5] L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl, “Impact of etcd deployment on Kubernetes, Istio, and application performance,” *Software: Practice and Experience*, vol. 50, no. 10, pp. 1986–2007, Aug. 2020, doi: https://doi.org/10.1002/spe.2885.
[6] L. Sun and D. Berg, “Istio explained getting started with service mesh.”
[7] W. Zhang, “Improving microservice reliability with istio,” 2020. https://willczhang.github.io/downloads/paper.pdf