

# Optimasi Penggunaan Resource Microservice berdasarkan Customer Behavior Model Graph (CBMG) dengan Horizontal Pod Autoscaling (HPA) = Optimizing Microservice Resource Usage based on Customer Behavior Model Graph (CBMG) with Horizontal Pod Autoscaling (HPA)

Binar Qalbu Cimuema, author

Deskripsi Lengkap: <https://lib.ui.ac.id/detail?id=9999920542291&lokasi=lokal>

---

## Abstrak

Berbeda dengan arsitektur *monolith*, arsitektur *microservice* dapat melakukan *scaling* secara independen pada *service* tertentu saja, memberikan fleksibilitas yang lebih besar dalam menanggapi lonjakan *traffic* dan tentunya lebih menghemat *resource* dibanding *monolith*. Tidak semua *service* pada suatu aplikasi perlu dilakukan *scaling*, hanya *service* dengan *load processing* tertinggi saat menerima banyak *request* yang perlu dilakukan *horizontal* *scaling* untuk menghemat *resource*. Tetapi penentuan *service* yang harus dilakukan *scaling* harus dilakukan secara benar agar sesuai dengan kebutuhan pengguna. Salah satu metode yang bisa digunakan adalah *Customer Behavior Model Graph* (CBMG) dengan melihat probabilitas perpindahan halaman yang dilakukan oleh pengguna. Dari metode tersebut dapat ditemukan halaman yang paling sering diakses oleh pengguna sebelum akhirnya ditentukan *service* dengan *load processing* tertinggi. Salah satu teknik yang dimiliki oleh kubernetes adalah *Horizontal Pod Autoscaling* (HPA) yang memungkinkan untuk melakukan *scaling* hanya pada salah satu *pod*. Pada kubernetes, *service* lebih dikenal sebagai *pod*. Dari pengimplementasian HPA didapatkan bahwa pada percobaan terjadi penurunan *access failure rate* dari sebelum implementasi sebesar 17.19% dan 20.52% dan setelah implementasi turun menjadi 4.86% dan 5.44%. Selain itu terdapat kenaikan *throughput* pada percobaan dari sebelum implementasi sebesar 25.00 *request*/detik dan 41.30 *request*/detik, setelah implementasi didapatkan sebesar 39.30 *request*/detik dan 51.10 *request*/detik. Pada percobaan lainnya didapatkan sebelum implementasi sebesar 4.60 *request*/detik dan 4.20 *request*/detik, setelah implementasi didapatkan sebesar 15.50 *request*/detik dan 13.80 *request*/detik. Dari hasil implementasi bisa dilihat bahwa melakukan peningkatan pada salah satu *pod* sudah cukup untuk meningkatkan kinerja aplikasi website dengan *resource* yang tersedia dan dapat dioptimalkan dengan maksimal. Implementasi dilakukan pada salah satu aplikasi website *microservice* *teastore*, dengan strategi *scaling* berdasarkan CMBG, optimasi yg dilakukan berhasil menurunkan *access failure rate* dan meningkatkan *throughput*, meskipun menggunakan jumlah *resource* yang sama. dengan kata lain, setelah strategi yang dirancang diimplementasikan, penggunaan *resource* menjadi lebih optimal untuk melayani request-request yang ada.

.....Unlike monolithic architecture, *microservice* architecture can independently scale specific *service*s, providing greater flexibility in responding to traffic spikes and, of course, saving more *resource*s compared to monoliths. Not all *service*s in an application need to be

scaled; only pods with the highest load processing when receiving many `request`s need to be horizontally scaled to save `resource`s. However, determining which `service`s need scaling must be done properly to meet `user` needs. One method that can be used is the Customer Behavior Model Graph (CBMG), which looks at the probability of `user` page transitions. From this method, the most frequently accessed pages by `user`s can be identified before determining the `service` with the highest load processing. One technique available in Kubernetes is `Horizontal Pod Autoscaling` (HPA), which allows scaling to be done only on specific pods. From the implementation of HPA, it was found that there was a decrease in the `access failure rate` from before implementation by 17.19% and 20.52%, and after implementation, it decreased to 4.86% and 5.44%. Additionally, there was an increase in throughput from before implementation by 25.00 `request`/second and 41.30 `request`/second, after implementation, it was found to be 39.30 `request`/second and 51.10 `request`/second. In another experiment, before implementation was 4.60 `request`/second and 4.20 `request`/second, after implementation, it was 15.50 `request`/second and 13.80 `request`/second. Improving the performance of one pod is sufficient to enhance the performance of the website application with the available `resource`s and can be optimized to the maximum. The implementation was carried out on one `microservice` website application, making it better than monolithic architecture, which needs to scale the entire application.