Jurnal Teknik Informatika (JUTIF)
Vol. 3, No. 4, Agustus 2022, hlm. 923-927

DOI: https://doi.org/10.20884/1.jutif.2022.3.4.484
p-ISSN: 2723-3863
e-ISSN: 2723-3871

AUTO SCALING DATABASE SERVICE WITH MICRO KUBERNETES CLUSTER

Anita Rosdina Nasution[*1], Favian Dewanta[2], Bagus Aditya[3]

[1,2,3]Fakultas Teknik Elektro, Universitas Telkom, Indonesia
Email: [1]anitanasution@student.telkomuniversity.ac.id, [2]favian@telkomuniversity.ac.id,
[3]goesaditya@telkomuniversity.ac.id

(Naskah masuk: 20 Juli 2022, Revisi : 27 Juli 2022, diterbitkan: 20 Agustus 2022)

Abstract

Data storage media, or what is often referred to as a database is something that is quite vital for technological developments. As the number of transaction and user activities increase, then the size of the database will increase which allows database services to experience downtime due to server maintenance and limited computing resources For that, a server system needs an infrastructure that can replicate itself, so that it will avoid downtime. This infrastructure can be built using a container orchestration tool called Kubernetes which has high availability and autoscaler features, so it can replicate and guarantee service availability, to avoid downtime. This research builds a MongoDB NoSQL database service. This service is built using micro Kubernetes clusters from several different data centers. This service also implements a horizontal pod autoscaler feature that is capable of replicating pods, to increase high availability and avoid downtime. The autoscaling process will be tested by providing a load request for the service. Testing is done several times on each server. This study will compare the MongoDB service that was built monolithically with a micro Kubernetes cluster, and with HPA features and without HPA features by paying attention to several things. Based on Response Time, Response Code per Seconds, and CPU Usage, the results shows that the service built on a micro Kubernetes cluster with HPA features is the best, with a constant response time value below 100 ms, Response Code per Seconds reaches 500 threads per second. seconds, and CPU Usage in the range of 30 – 55%.

Keywords: *HPA, Micro Kubernetes Cluster, MongoDB.*

LAYANAN DATABASE DENGAN PENGSKALAAN OTOMATIS MENGGUNAKAN MICRO KUBERNETES CLUSTER

Abstrak

Media penyimpanan data, atau yang sering disebut dengan *database* merupakan hal yang cukup vital bagi perkembangan teknologi. Seiring dengan bertambahnya jumlah transaksi dan aktivitas pengguna, maka ukuran database akan meningkat yang memungkinkan layanan database mengalami *downtime* dikarenakan adanya perawatan server serta sumber daya komputasi yang terbatas. Untuk itu, sebuah sistem server membutuhkan infrastruktur yang dapat melakukan replikasi atas dirinya, sehingga akan menghindari terjadinya *downtime*. Infrastruktur tersebut dibangun dengan menggunakan sebuah *container orchestration tool* bernama micro Kubernetes yang memiliki fitur *high availability* dan *autoscaler*, sehingga dapat melakukan replikasi dan dapat menjamin ketersediaan layanan. Penelitian ini bertujuan untuk membangun sebuah layanan *database* NoSQL, yaitu MongoDB. Layanan ini dibangun dengan menggunakan *micro Kubernetes cluster* dari beberapa *data center* yang berbeda. Layanan mengimplementasikan fitur *horizontal pod autoscaler* yang mampu melakukan replikasi *pods*, untuk meningkatkan *high availability* sehingga dapat menghindari *downtime*. Proses *autoscaling* pada layanan akan diuji dengan mengirimkan *load request*. Pengujian dilakukan beberapa kali terhadap setiap *server*nya. Penelitian ini akan membandingkan *service* MongoDB yang dibangun secara monolitik dengan *micro Kubernetes cluster*, dan dengan fitur HPA dan tanpa fitur HPA dengan memperhatikan parameter. Berdasarkan *Response Time, Response Code per Seconds*, dan *CPU Usage*, maka hasil penelitian menunjukkan *server* yang dibangun di atas *micro Kubernetes cluster* dengan fitur HPA adalah yang terbaik, dengan nilai *response time* yang konstan di bawah 100 ms, *Response Code per Seconds* mencapai 500 *thread per seconds*, dan *CPU Usage* pada rentang 30 – 55%.

Kata kunci: *HPA, Micro Kubernetes Cluster, MongoDB.*

1. PENDAHULUAN

Meningkatnya pengguna layanan mengakibatkan banyak sekali data yang perlu diolah oleh pihak penyedia layanan, sehingga perlu memiliki kemampuan pengolahan *database* yang baik, agar data yang dimiliki pelanggan tersimpan dengan baik, dan tidak terjadi kesalahan penyusunan informasi. Penggunaan *database* jenis NoSQL bisa menjadi solusi untuk masalah ini. NoSQL didesain untuk mengolah data lalu menyimpannya dengan konsisten, dengan tujuan bisa dilihat lagi dikemudian hari oleh pelanggan juga penyedia layanan. NoSQL dikembangkan untuk menyelesaikan masalah *scaling* dan *reliability*, dan sangat cocok untuk aplikasi yang berkembang dengan cepat, karena bersifat *dynamic schema*.

Semakin banyak pengguna layanan, maka *resource* dan *server* perlahan akan habis. Penyedia layanan akan kewalahan mengerjakan permintaan pelanggan, akibatnya terjadi *downtime* pada layanan. Salah satu cara untuk menghindari kasus ini, dapat digunakan Docker sebagai *container application*, yang membuat *High Availability* sehingga layanan database tetap terjaga. *High Availability* adalah kemampuan sistem atau *cluster* untuk menjaga layanannya tetap bekerja dengan baik. Berfungsi untuk mengurangi *error* pada layanan, ketika layanan sedang berjalan.

Penelitian sebelumnya pernah membahas tentang penggunaan metode load balancing untuk meningkatkan ketersediaan layanan dengan cara pembagian beban terhadap worker node. Salah satunya adalah Penggunaaan Algoritma WRR pada LVS/TUN dengan hasil bahwa kapasitas transaksi setiap detik dari *single database server* dengan dua buah *database server* yang terkluster dapat meningkat sebanyak 50%[1]. Penelitian lainnya adalah Implementasi Haproxy sebagai *Load Balancer Web Server*. Penelitian ini membuktikan jika sistem *load balancing* dapat bekerja dengan baik ketika *request* datang dari client telah berhasil 2 didistribusikan oleh balancer kepada setiap *node cluster*, sehingga server tidak mengalami *overload*[2].

Pada penelitian ini, penulis akan membangun sebuah *server database* MongoDB menggunakan *micro Kubernetes cluster* yang memiliki fitur *Horizonal Pod Autoscaler* yang mampu menduplikasi *pod*, sehingga dapat menjamin seluruh request yang masuk dapat ditanggapi oleh *server*. *Micro Kubernetes Cluster* juga memiliki fitur *Load Balancer* yang akan membagi *request* kepada *node* yang tergabung di *cluster*, untuk menjaga *server* tidak mengalami *downtime*[3].

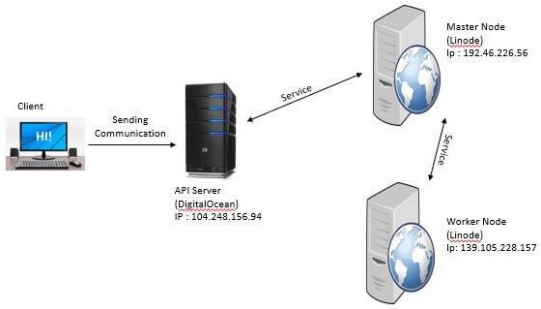
Tujuan dari penelitian ini adalah untuk mengetahui bagaimana pengaruh *micro Kubernetes cluster* dengan *high availability* yang tinggi untuk suatu infrastruktur database NoSQL dan untuk mengetahui bagaimana perbandingan *CPU Usage*, *Response Code per Seconds* dan *Response Time* dari *service* yang dibangun dengan *micro Kubernetes cluster* dengan *server* yang dibangun secara *monolitik.*

2. METODE PENELITIAN

Pendekatan dan metode yang digunakan dalam penyelesaian tugas akhir ini adalah :

1. Perancangan Sistem

Membuat rancangan sistem dan scenario yang akan diterapakan pada penelitian. Penulis menggunakan beberapa *server*, yang digunakan sebagai *server* monolitik, *master node* untuk *micro Kubernetes cluster*, *worker node micro Kubernetes cluster* dan satu *server* sebagai API penghubung, agar kita dapat langsung terhubung dengan *server database* yang dibangun. *Server* API dapat terhubung dengan *server* monolitik dan *master node dari micro Kubernetes cluster*.



Gambar 1. Physical Architecture

Terdapat dua buah cloud server yang terletak secara fisik pada dua buah data center di negara yang berbeda. Kedua cloud server ini telah dikonfigurasi masing-masing dan menjadi satu buah cluster microservice menggunakan arsitektur microcluster Kubernetes. Masing masing cloud server memiliki satu buah nodes. Nodes pada cloud server A berperan sebagai node utama sedangkan node yang ada di cloud server B berperan sebagai back up node. Node utama menjalankan service mongoDB sedangkan back up node dalam status aktif tanpa menjalankan service apapun. Ketika Node utama pada server A tidak dapat lagi menampung workload dengan cara mengaktifkan salah satu objeknya yaitu Horizontal Pod Autoscaler. Dengan fitur ini, maka Kubernetes akan menduplikasi service MongoDB yang berasal dari server A secara otomatis dan workload ini akan dilimpahkan kepada node back up yang ada di cloud server B.

2. Implementasi dan Pengukuran

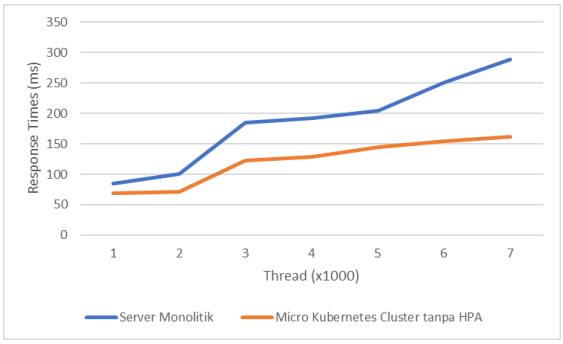
Mengimplementasikan rangcangan sistem yang dibuat, selanjutkan melakukan pengukuran parameter yang akan diteliti. Setelah seluruh *server* selesai diimplementasi, maka akan dilakukan pengujian terhadap setiap *server*nya. Pengujian dengan memberikan *load request* kepada *server*. Setiap *server* diuji dengan 1000, 2000, 3000, 4000, 5000, 6000, dan 7000 *thread* dengan dilakukan 5 kali pengujian untuk setiap *thread*. Dilakukan analisis

seteleah didapatkan hasil dari pengukuran. Parameter yang dianalisis adalah parametes *response time*, *CPU Usage*, dan *Response code per seconds*.

3. HASIL DAN PEMBAHASAN

3.1. Perbandingan Service Mongodb Dibangun Secara Monolitik Dengan Dibangun Micro Kubernetes Cluster

3.1.1. Perbandingan Berdasarkan *Response Time*

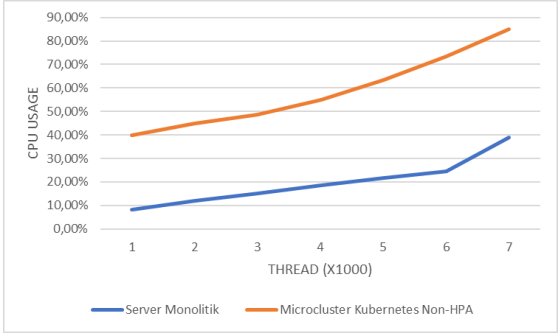


Gambar 2. Perbandingan *Response Time Server* Monolitik dan *Micro Kubernetes cluster* tanpa HPA

Pengujian dilakukan sebanyak tujuh kali untuk setiap *server*, dari 1000 *thread* sampai 7000 *thread* (kelipatan 1000), dengan spesifikasi *server* yang sama. Dari grafik dapat dilihat jika nilai dari *response time service* yang dibangun dengan *microcluster* Kubernetes lebih rendah dibandingkan *service* yang dibangun secara monolitik. Ketika dilakukan pengujian sebesar 7000 *thread*, *server micro Kubernetes cluster* dapat me*response request* dalam waktu 150 ms, dan *server* monolitik dengan waktu 288 ms.

Gambar 2 menunjukkan bahwa *service* yang dibagun dengan menggunakan *micro Kubernetes server* lebih baik dibandingkan dengan *server* yang dibangun secara monolitik. Karena, *response time* dari *server* yang dibangun dengan *micro Kubernetes cluster* lebih rendah daripada *service* monolitik. Artinya, *server micro Kubernetes cluster* dapat lebih cepat meanggapi *request* yang diberikan.

3.1.2. Perbandingan berdasarkan CPU Usage

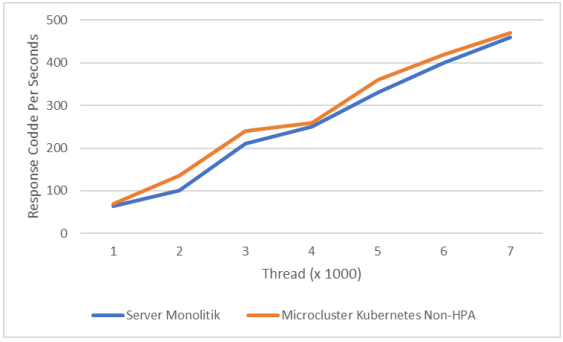


Gambar 3. Perbandingan *CPU Usage Server* Monolitik dan *Micro Kubernetes Cluster* tanpa HPA

Dari grafik yang ditampilkan pada Gambar 3 dapat dilihat jika *CPU Usage* selalu naik berbanding lurus dengan jumlah *thread* yang dikirimkan. Penggunaan CPU service yang dibangun secara monolitik ketika diserang sebanyak 7000 *thread* menggunakan JMeter mencapai angka 40%. Penggunaan *CPU service* yang dibangun dengan *microcluster Kubernetes* tertinggi mencapai angka 85,10%. Hal ini menunjukkan bahwa *micro Kubernetes cluster* memiliki beban komputansi yang lebih berat dibandingkan *server* monolitik. Ini terjadi akibat pada *server* monolitik hanya menjalankan *basic service* mongoDB saja, sedangkan pada *server micro Kubernetes cluster* berjalan beberapa *service*, seperti microk8s, kubectl, docker, PV, PVC, NodePort SVC, dan HPA.

Kelebihan dari *server micro Kubernetes cluster* dapat dilihat dari naiknya *CPU Usage*. Kenaikan *CPU Usage Micro Kubernetes cluster* naik secara eksponensial, berbeda dengan grafik *server* monolitik yang menaik tajam. CPU pada *micro Kubernetes cluster* akan kembali turun pada saat node lain aktif untuk membantu *node master* melayani *request*. Berbeda dengan *server* monolitik yang akan selalu tinggi jika mencapai puncak utilisasi CPU dan berakibat semua layanan berhenti dan *server* akan mengalami *downtime*.

3.1.3. Perbandingan Berdasarkan *Response Code Per Seconds*



Gambar 4. Perbandingan *Response Code Server* Monolitik dan *Micro Kubernetes Cluster* tanpa HPA

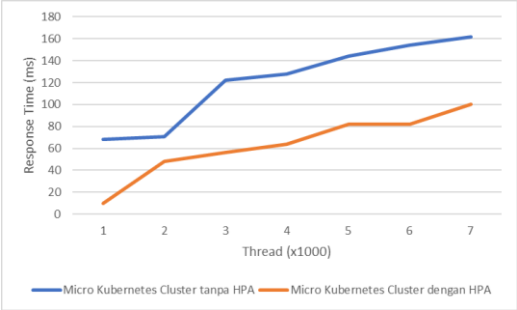
Response Code adalah jumlah *code* yang berhasil diterima oleh *service* ketika *thread* dikirim oleh aplikasi JMeter. Dapat dilihat pada grafik, jika jumlah *thread* yang berhasil diterima oleh *server* yang dibangun dengan *Micro Kubernetes Cluster* selalu lebih banyak dibandingkan dengan *server* monolitik. *Server* monolitik berhasil menerima *request* pada rentang 65 – 460 *thread per seconds*, sedangkan *server micro Kubernetes cluster* tanpa HPA menerima *request* pada rentang 70 – 470 *thread per seconds*.

Gambar 4 menunjukkan jika *server* yang dibangun dengan *micro Kubernetes cluster* lebih baik dibandingkan *server* monolitik. Hal ini ditandai dengan jumlah *code* yang berhasil diterima oleh

micro Kubernetes cluster selalu lebih banyak dibandingkan *server* monolitik.

3.2. Perbandingan Micro Kubernetes Cluster Tanpa HPA Dengan Micro Kubernetes Cluster Dengan HPA

3.2.1. Perbandingan Berdasarkan *Response Time*

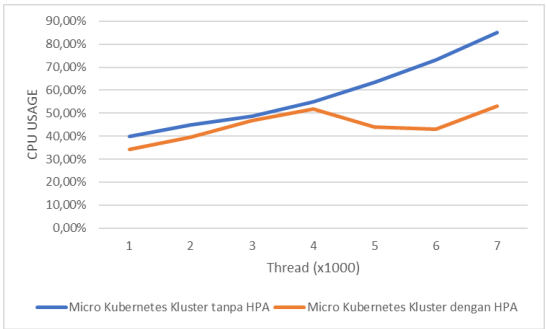


Gambar 5. Perbandingan *Response Time Service* HPA dan tanpa HPA

Kedua *service* diuji sebanyak tujuh kali setiap servicenya, yaitu 1000, 2000, 3000, 4000, 5000, 6000, dan 7000 *thread* menggunakan *tool* Apache JMeter. Seperti yang dapat dilihat dari grafik, *service* yang dibangun dengan *microcluster Kubernetes* dengan HPA memiliki nilai *response time* yang lebih rendah dibandingkan dengan tanpa HPA. *Service* yang memiliki fitur *Horizontal Pod Autoscaler* dapat me*response request* dalam waktu 80 ms ketika di *hit* dengan 7000 *thread*, dan *server* yang tidak memiliki HPA merespons dalam waktu 140 ms.

Dari Gambar 5, menunjukkan bahwa *server* yang dibangun dengan *micro Kubernetes cluster* dengan fitur *Horizontal Pod Autoscaler* lebih baik dibandingkan *server* tanpa fitut *Horizontal Pod Autoscaler*. Hal ini dibuktikan degan nilai dari *response time server* dengan HPA selalu lebih rendah dibandingkan *server* tanpa HPA. Ini menunjukkan bahwa *server* dengan HPA dapat memproses *request* lebih cepat dibandingkan *server* tanpa fitur HPA.

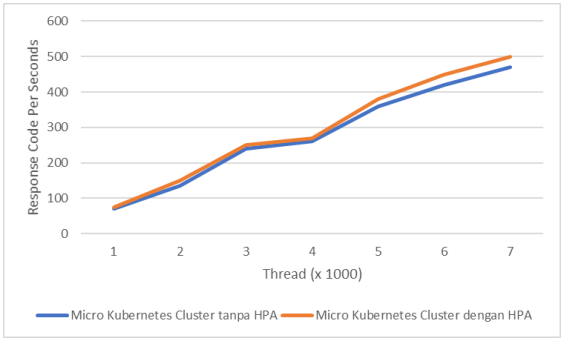
3.2.2. Perbandingan Berdasarkan *CPU Usage*



Gambar 6. Perbandingan *CPU Usage* HPA dan tanpa HPA

Gambar 6 merupakan grafik perbandingan dari hasil pengujian server tanpa HPA dan dengan HPA. Pengujian tanpa HPA memiliki nilai antara 40- 85%. Sedangkan nilai pengujian dengan HPA memiliki nilai rentang yang lebih rendah, yakni 30-55%. Nilai pengujian tanpa HPA cenderung naik, sedangkan pengujian dengan HPA naik pada jumlah *threads* 1000-4000 dan turun ketika *thread* mencapai 4000-7000. Penurunan *CPU Usage* pada *server Micro Kubernetes Cluster* dengan HPA merupakan akibat dari berjalannya fitur *Load Balancing*, yaitu terbentuknya *pod* replika untuk membantu *master pod* dalam menanggapi *request*. Sehingga, penggunaan CPU pada *master node* berkurang, karena Sebagian *thread* sudah dibagi pada *worker node*. Maka dapat ditarik kesimpulan bahwa *server* yang dibangun dengan *micro Kubernetes cluster* dengan HPA lebih baik dibandingkan *server* tanpa HPA. Karena nilai dari *CPU Usage* selalu lebih rendah dibandingkan *server* tanpa HPA.

3.2.3. Perbandingan Berdasarkan *Response Code Per Seconds*



Gambar 7. Perbandingan *Response Code* HPA dan tanpa HPA

Response Code adalah jumlah *thread* yang berhasil diterima oleh *server* setelah dikirim dari aplikasi Apache JMeter. *Server Micro Kubernetes Cluster* dengan HPA menerima *request* pada rentang 75 – 500 *thread per seconds*, sedangkan *server Micro Kubernetes Cluster* tanpa HPA berhasil menerima *request* pada rentang 70 – 470 *thread per seconds*. Gambar 7 merupakan grafik perbandingan dari *response code per seconds server micro* kubernetes *cluster* tanpa HPA dan dengan HPA. Dari hasil analisis di atas, diambil kesimpulan jika *server* yang dibangun dengan *micro Kubernetes cluster* dengan HPA lebih baik jika dibandingkan dengan *micro Kubernetes cluster* tanpa HPA. Karena *server* dengan HPA mampu me*response code* lebih banyak dibandingkan dengan *server* tanpa HPA.

4. KESIMPULAN

Penggunaan *micro Kubernetes cluster* dapat membantu *server* untuk menghindari terjadinya *downtime*, akibat dari fitur *load balancing* yang dimiliki olehnya. Dari dua perbandingan yang dilakukan, didapatkan hasil yaitu *server* yang dibangun dengan *micro Kubernetes cluster* dengan fitur *horizontal pod autoscaler* menjadi *server*

terbaik. Hal ini dapat disimpulkan dari nilai *response time* yang selalu berada di bawah 100 ms, *response code per seconds* mencapai 500 *threads per seconds*, dan *CPU Usage* pada rentang 30 – 55.

Server micro kubernetes cluster dapat melakukan replikasi dalam waktu satu menit, dan *pod* replikasi akan hilang dalam waktu 6 menit setelah *pod* tidak menjalankan *request* lagi. Selama pengujian, tidak ada perubahan signifikan yang terjadi pada *memory usage*.

UCAPAN TERIMAKASIH

Peneliti berterima kasih kepada Telkom University yang telah bersedia untuk melakukan pembiayaan agar penelitian ini dapat berhasil.

DAFTAR PUSTAKA

[1] B. Aditya and T. Julhana, "A High Availability (HA) MariaDB Galera Cluster across data center with optimized WRR Scheduling Algorithm of LVS - TUN," 2015 9th International Conference on Telecommunication System Services and Application (TSSA), 2015, pp. 1-5, doi: 10.1109/TSSA.2015.7440452.

[2] A. Rahmatulloh, M. S. N. Firmansyah "Implementasi Load Balancing Web Server menggunakan Haproxy dan Sinkronisasi File pada Sistem Informasi Akademik Universitas Siliwangi," *Jurnal Nasional Teknologi & Sistem Informasi*. Vol 3, No 2, 2017.

[3] Anaqvi, " Introduction to MicroK8s " Sep, 16. 2019 Available : https://ubuntu.com/blog/introduction-to-microk8s-part-1-2.

[4] Indonesian Cloud, " Mengenal Cloud Computing: Pengertian, Tipe dan Fungsinya" Available : https://indonesiancloud.com/mengenal-cloudcomputing/

[5] V. G. da Silva, M. Kirikova, and G. Alksnis, "Containers for Virtualization: An Overview," Appl. Comput. Syst., vol. 23, no. 1, pp. 21–27, 2018, doi: 10.2478/acss-2018-0003

[6] X. L. Xie, P. Wang, and Q. Wang, "The performance analysis of Docker and rkt based on Kubernetes," ICNC-FSKD 2017 - 13th Int. Conf. Nat. Comput. Fuzzy Syst. Knowl. Discov., pp. 2137–2141, 2017, doi: 10.1109/FSKD.2017.8393101.

[7] F. K. R. Rasandriya, "Implementasi Platform Internet of Things (IoT) pada Virtualisasi Container Menggunakan Docker," Tugas Akhir, Telkom University, 2020.

[8] G. C. Platform and A. C. U. Docker, "Building Modern Clouds : Using Docker , Kubernetes," 2019 IEEE 9th Annu. Comput. Commun. Work. Conf., pp. 184–189, 2019, doi: 10.1109/CCWC.2019.8666479.

[10] C. -Chang, S. -R Yang, E. -H Yeh, P. Lin and J. -Y Jeng "A Kubernetes-Based Monitoring Platform for Dynamic Cloud Resource Provisioning," GLOBECOM 2017 - 2017 IEEE Global Communications Conference, 2017, pp. 1-6, doi : 10.1109/GLOCOM.2017.8254046.