Perancangan dan Analisis Elastic Cloud-Based Data processor menggunakan Kubernetes Event-Driven Autoscaler

1st Xavier Samuel MFP
Fakultas Informatika
S1 Teknologi Informasi
Bandung, Indonesia
xavier@student.telkomunivsity.ac.id

2nd , Siti Amatullah Karimah S.T., M.Kom.
Fakultas Informatika
S1 Teknologi Informasi
Bandung, Indonesia
karimahsiti@telkomuniversity.ac.id

3rd Ridha Muldina Negara, S.T., M.T.,
Fakultas Teknik Elektro
S1 Teknik Telekomunikasi
Bandung, Indonesia
ridhanegara@telkomuniversity.ac.id

Penelitian ini membahas perancangan dan analisis Elastic Cloud-Based Data processor yang memanfaatkan Kubernetes Event-Driven Autoscaler. Dalam konteks perkembangan sistem aplikasi berbasis event-driven yang semakin kompleks dan membutuhkan pengolahan data yang cepat serta skalabilitas yang dinamis, penggunaan teknologi cloud dan otomatisasi menjadi kunci utama. Penelitian ini menitikberatkan pada pengembangan prosesor data yang mampu mengelola beban kerja yang bervariasi dari sistem event-driven dengan menggunakan Kubernetes sebagai platform manajemen container dan Autoscaler yang responsif terhadap peristiwa (Event-driven). Analisis mendalam terhadap penggunaan Kubernetes Event-Driven Autoscaler diintegrasikan dengan infrastruktur cloud untuk memproses data dengan optimal. Metode eksperimental digunakan untuk menguji kinerja dari sistem yang diusulkan, termasuk pengujian terhadap skalabilitas dan efisiensi pemrosesan data. Penelitian ini menghasilkan analisis mengenai perilaku HPA dalam melakukan autoscaling data-processor pada sistem Event-Driven menggunakan KEDA (Kubernetes Event-Driven Autoscaler) dan membandingkan dengan metode scaling yang sudah. Penelitian ini juga memparkan efisiensi KEDA dalam melakukan autoscaling pada data processor

***Kata Kunci: Cloud, Kubernetes, Event-Driven, Scalability*.**

I. PENDAHULUAN

Sistem Event-Driven seperti IoT mengandalkan pertukaran informasi secara real-time dan efisien antara berbagai perangkat dan cloud, yang sering kali membutuhkan layanan data processor yang optimal agar para pemangku kepentingan tetap mendapatkan informasi. Message broker biasanya digunakan untuk memfasilitasi komunikasi dalam sistem Event-Driven, tetapi ada berberapa permasalahan. Penggunaan Message Broker terkadang bisa jadi sulit, karena kompleksitas yang terkait dengan penanganan data, skalabilitas, dan manajemen sumber daya. Salah satu tantangan yang sering muncul dalam sistem berbasis IoT adalah pengelolaan antrian (queue) atau pesan (messsage) dalam message broker. Antrian (Queue) ini sering kali menumpuk sejumlah pesan yang cukup besar, yang menyebabkan inefisiensi dan potensi kemacetan data atau pesan yang berada pada message broker. Message dapat tidak diproses karena keterbatasan sumber daya yang tersedia. Situasi ini dapat memiliki konsekuensi yang luas, termasuk pemberitahuan yang tertunda, kehilangan data, dan, dalam kasus yang ekstrim, sistem memasuki kondisi pemblokiran atau non-pemrosesan [14].

Untuk mengatasi tantangan ini, penulis mengusulkan solusi inovatif yang bertujuan untuk mengoptimalkan kinerja layanan pemrosesan data dalam sistem berbasis IoT. Solusi ini memanfaatkan Horizontal Pod Autoscaler (HPA) pada Kubernetes yang merupakan platform orkestrasi kontainer merupakan salah satu teknologi cloud-native yang mendukung High Availability (HA) dan Elastisitas [14]. Penilitian kali ini penulis menggunakan Kubernetes Event-Driven Autoscaler (KEDA) untuk memastikan antrean pesan dikelola dan diproses secara efisien. Dengan demikian, penulis bertujuan untuk meningkatkan keandalan dan daya tanggap sistem IoT sekaligus memitigasi masalah yang terkait dengan antrean pesan yang besar dan tidak diproses serta keterbatasan sumber daya. Solusi ini menyesuaikan alokasi sumber daya secara dinamis berdasarkan beban sistem, sehingga memungkinkan pemrosesan pesan yang efisien dan mencegah sistem memasuki kondisi pemblokiran atau non-pemrosesan.

Sebelumnya sudah dilakukan penelitian terkait dengan Horizontal Pod Autoscaler (HPA) dengan membandingkan metric terbaik dalam melakukan autoscaling menggunakan HPA yaitu dengan membandingkan metric Kubernetes Resource Metric (KRM) dan Prometheus Custom Metric (PCM) dan mengevaluasi perilaku HPA berdasarkan kedua metric tersebut [15]. Penelitian selanjutnya membahas mengenai pengenalan HPA pada single-board computer untuk nantinya dapat diimplementasikan pada IoT [14]. Kedua penelitian diatas masih membahas HPA dengan metric yang diambil dari CPU dan Memory hal ini menjadi masalah pada kasus tertentu terutama dalam hal responsivitas dikarenakan HPA hanya scaling berdasarkan kedua metric tersebut sedangkan dalam teknologi IoT banyak faktor yang memungkinkan suatu sistem terjadi lambat atau ter-blok seperti penumpukan message pada message queue, maka dari itu penelitian kali penulis mengimplementasikan HPA mennggunakan metric yang lebih luas menggunakan Kubernetes Event-Driven Autosclaer (KEDA) yang scaling suatu Pod dengan metric yang diambil dari message broker.

II. KAJIAN TEORI

A. Cloud Computing

National Institute of Standards and Technology (NIST) mendefinisikan bahwa "Komputasi awan adalah sebuah model untuk memungkinkan akses jaringan yang mudah, dapat diakses di mana saja, sesuai permintaan, ke dalam sebuah kumpulan sumber daya komputasi yang dapat dikonfigurasi (misalnya, jaringan, server, penyimpanan, aplikasi, dan layanan) yang dapat dijalankan dan dihentikan dengan cepat dengan pengelolaan yang minimal.",Cloud computing merupakan sumber daya komputasi yang dapat diakses melalui internet.

Cloud Computing adalah suatu Integrasi dari Komputasi Terdistribusi, Komputasi Pararel, Komputasi Terutilisasi, virtualisasi, Load Balancing. Pada era modern Cloud Computing telah mendukung berbagai kegunaan dibandingkan dengan sistem tradisional (sistem komputasi fisik) dimana pada sistem tradisional jumlah transaksi cukup terbatas karena keterbatasan sumber daya. Cloud Computing dapat digunakan dalam berbagai praktik komputasi seperti pre-processing, analysis, dan memprediksi suatu kejadian [2].

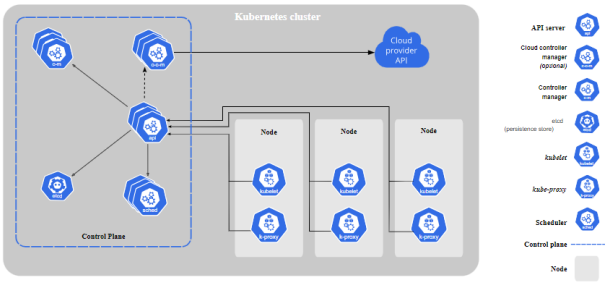
B. Data Processor

Data processor merupakan suatu komponen pada platform yang melakukan consume message dari Kafka queue, selanjutnya data prosesor melakukan berbagai tindakan terhadap payload yang telah didapat dari Kafka queue seperti menyimpan ke database, mengirim notifikasi, mengirim email dan pengelolaan data lainnya. Data processor memantau dan memproses message yang masuk satu per satu [16].

C. Kubernetes

Kuberentes merupakan platform untuk mengelola layanan berbasis container. Kubernetes pada umumnya merupakan suatu container orchestrator yang memfasilitasi deployment terotomatisasi, High Availability, Load Balancing, Service Discovery, Self-Healing, Horizontal Scaling dan lain-lain.

Cluster Kubernetes terdiri dari sekumpulan mesin worker yang disebut nodes, yang menjalankan aplikasi terkontainerisasi. Setiap cluster setidaknya mempunyai satu worker node. Worker node menjalankan Pod yang merupakan suatu komponen untuk menjalankan aplikasi. Control plane atau master node mengelola node-node worker dan Pod pada cluster. Gambar II.1 merupakan gambarn arsitektur pada Kubernetes cluster beserta komponen yang terdapat di dalamnya [22].



Gambar II.1 Arsitektur Kubernetes

D. Apache Kafka

Apache kafka merupakan suatu data streaming platform yang mirip dengan message queue. Message dapat dipublish dan consume dari beberapa topik sehingga kitab isa membedakan setiap message yang dihasilkan dari beberapa node. Topik pada kafka bisa memiliki partisi atau queues. Message dengan topik tertentu di alokasikan secara round-robin ke beberapa partisi. Ketika beberapa consumer memproses message dari suatu topik pada saat yang sama, setiap partisi diconsume oleh consumer yang berbeda-beda[16].

E. Horizontal Pod Autoscaler (HPA)

Pada Kubernetes, HPA merupakan salah satu fitur unggulan yang dapat melakukan penambahan dan pengurangan pod secara otomatis tanpa membuat aplikasi yang dijalankan pada pod berhenti atau mati. Setelah pod baru telah berhasil diskalakan maka pod baru tersebut membagi beban kerja yang datang ke pod lainnya. Secara default setiap 15 detik, juga disebut sebagai sync cycle, kube-controller-manager melakukan komparasi dari metric yang telah dikumpulkan terhadap ambang batas yang telah di konfigurasi pada HPA. Jumlah dari desired pod replicas (DPR) dikalkulasi dengan rumus pada Persamaan 1 [16]:

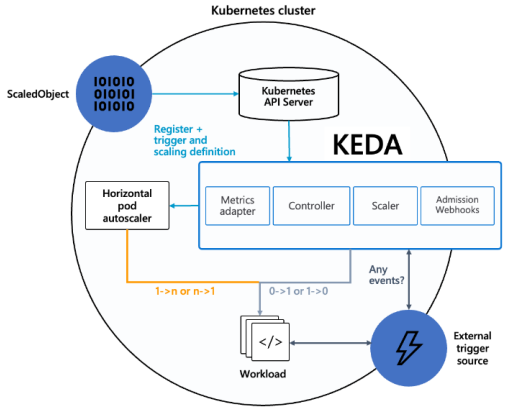
$$desiredReplicas = \left\lceil currentReplicas * \frac{currentMetricValue}{desiredMetricValue} \right\rceil$$

Persamaan 1 HPA Desired Replica

F. Kubernetes Event-Driven Autoscaler (KEDA)

KEDA (*Kubernetes Event Driven Autoscaling*) meropidan CRD atau custom resource definition dari *Kubernetes* yang berguna untuk menskalakan sumber daya *Kubernetes* secara otomatis berbasis sumber *Event*. *KEDA* di rancang untuk membantu aplikasi atau beban kerja merespon suatu *Event* dari Azure Queue, *Kafka* topic, RabbitMQ queue dan sejenisnya secara dinamis.

Dengan memanfaatkan *KEDA*, penskalaan aplikasi dapat dilakukan berdasarkan *metric* yang diambil dari sumber *Event*. *KEDA* juga memanfaatkan HPA pada *Kubernetes* untuk penskalaan secara otomatis berdasarkan *metric* tertentu. Sebagai contoh, jika suatu queue mulai penuh, *KEDA* secara otomatis menambah *pod* untuk mengangani beban kerja yang semakin tinggi, dan ketika suatu queue kosong maka *KEDA* melakakukan *scale down*. *KEDA* juga merupakan komponen ringan yang dapat diinstalasi pada berbagai *Kubernetes cluster*. *KEDA* berjalan bersama komponen kuberentes lainnya seperti HPA dan fungsionalitas tambahan tanpa menduplikasi yang sudah ada [23].



Gambar II.2 Arsitektur KEDA

G. Prometheus Metric Server

Sda Prometheus merupakan software monitoring yang mengekspos target yang dimonitor dengan endpoint dan secara periodik menarik *metric* melalui HTTP server. Prometheus dapat monitor berbagai macam target seperti *node*s, *pod*s, services, atau bahkan Prometheus sendiri. Operasi monitoring untuk setiap target disebut job [15].

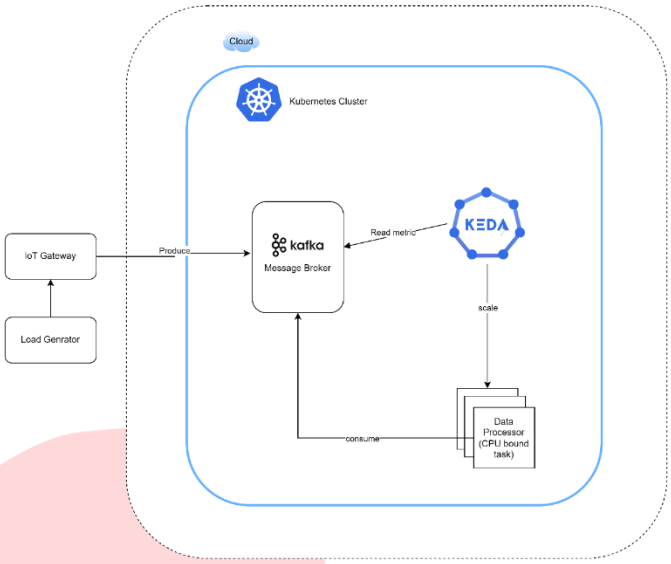
III. METODE

A. Arsitektur Sistem

Pada penelitian ini, pengujian dilakukan dengan dua metode autoscaling yang berbeda untuk mengevaluasi kinerja dan efektivitas masing-masing metode dalam mengelola skala aplikasi pada lingkungan Kubernetes. Pengujian pertama dilakukan menggunakan KEDA (Kubernetes Event-Driven Autoscaling), yang merupakan alat autoscaling berbasis Event yang dapat merespons berbagai jenis metrik seperti antrian pesan, penggunaan CPU, dan penggunaan memori untuk menentukan kapan harus meningkatkan atau mengurangi skala aplikasi..

Metode kedua yang diuji adalah PCM (Prometheus Custom Metric) sebagai metric server dari HPA (Horizontal Pod Autoscaler). PCM memungkinkan pengumpulan metrik kustom yang dikonfigurasi dalam Prometheus, yang kemudian digunakan oleh HPA untuk membuat keputusan autoscaling berdasarkan metrik-metrik tersebut. Dalam pengujian ini, berbagai skenario beban kerja disimulasikan untuk mengamati bagaimana kedua metode ini menangani perubahan beban dan memastikan aplikasi tetap responsif dan efisien dalam penggunaan sumber daya.

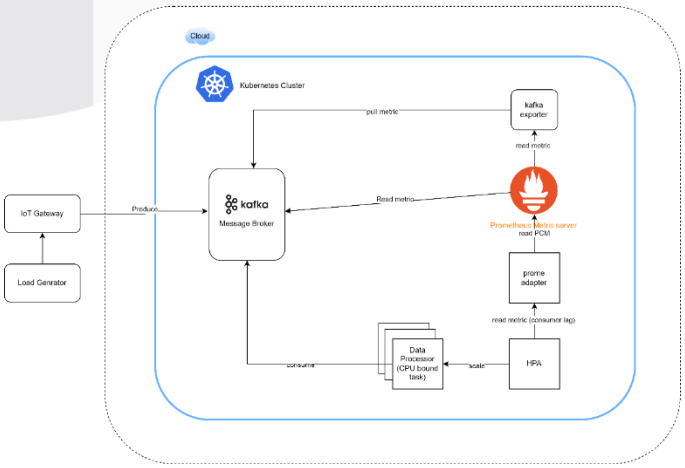
a. KEDA



Gambar III.1 Arsitektur HPA KEDA

Arsitektur pertama yang digunakan adalah metode autoscaling menggunakan KEDA (Kubernetes Event-Driven Autoscaler). Terdapat Iot gateway yang berperan sebagai pengirim data dari sensor IoT, pada penelitian ini Iot gateway menggunakan VM dummy dengan spesifikasi yang sudah disesuaikan dengan raspberry PI. Selanjutnya Iot gateway mengirim data denga waktu tertentu untuk menguji sistem autoscaling. Selanjutnya adalah apache kafka yang berperan sebagai message broker dari sistem berbasis Event-Driven dan sebagai tempat pertukaran data antara Iot gateway dan data processor. Komponen selanjutnya yaitu KEDA yang berperan sebagai autoscaler data processor. KEDA memiliki metric server bawaan yang berfungsi untuk mengambil metric dari berbagai sumber sehingga KEDA tidak memerlukan metric server terpisah agar dapat melaukakn autoscaling. Komponen terakhir yaitu data processor yang berperan sebagai pemroses data yang dikirim dari Iot gateway. Data processor dimana suatu layanan yang digunakan untuk beberapa hal seperti mengirim data menuju database, notifikasi, mengirim email, dan pengelolaan data lainnya.

b. PCM



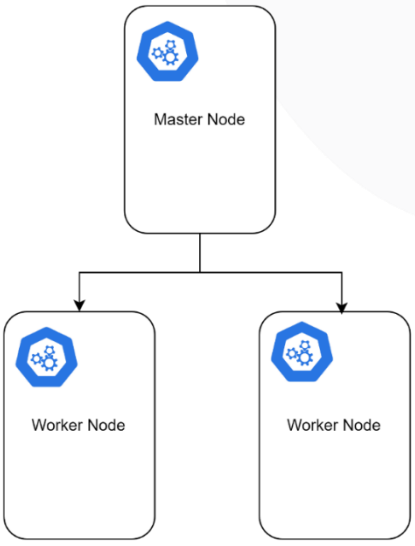
Gambar III.2 Arsitektur HPA PCM

Arsitektur kedua yaitu metode *autoscaling* menggunakan *PCM*. Metode *autoscaling* ini memanfaatkan prometheus custom *metric* sebagai *metric* servernya. Ada beberapa komponen yang berbeda pada arsitektur *KEDA*. Dibandingakan *KEDA* arsitektur ini lebih manual karena harus mengimplementasikan komponen penunjang *autoscaling* secara manual seperti exporter, *metric* server, dan adapter. *Kafka* exporter berfungsi untuk mengeksport *metric* dari *kafka* agar dapat diekspos oleh prometheus. Setelah itu prometheus mengambil *metric* dari *kafka* melalui *kafka* exporter yang sudah dideploy. Komponen selanjutnya yaitu prometheus adapter, HPA pada *Kubernetes* secara default tidak dapat mengambil *metric* dari *kafka* dan hanya dapat membaca *metric* dari kube-state-*metric* suatu *metric* default *Kubernetes* yang hanya membaca CPU dan Memory. Oleh karena itu agar HPA dapat mengambil *metric* dari *kafka*, HPA harus membutuhkan adapter yang berfungsi untuk mentranslasikan *metric* yang diambil oleh prometheus server

B. Spesifikasi

Pada Gambar III.3 dapat dilihat bahwa cluster yang digunakan pada penelitian ini terdiri dari satu node master dan 2 node worker yang nantinya digunakan untuk melakukan deployment layanan dan komponen yang diperlukan untuk kebutuhan penelitian. Penggunaan satu master dan 2 worker node pada Kubernetes cluster ini bertujuan untuk meningkatkan kemampuan High Availability dan elasticity.

Optimalisasi resource sangat penting pada lingkungan dengan keterbatasan resource seperti OpenStack cloud. Alokasi node yang lebih banyak dapat menyebabkan bottleneck pada cluster karena keterbatasan resource. Dengan mengurangi jumlah node, resource yang ada dapat dioptimalkan untuk mendukung performa cluster secara keseluruhan. Selain itu, memiliki 1 master node dan 2 worker node memungkinkan pemanfaatan maksimal dari resource yang ada tanpa mengorbankan kinerja atau stabilitas cluster. Hal ini penting agar eksperimen dan pengujian yang dilakukan dalam penelitian ini dapat berjalan dengan lancar tanpa kendala teknis yang berarti.



Gambar III.3 Kubernetes Node

Adapun cluster Kubernetes yang di deploy memiliki beberapa spesifikasi berbeda pada setiap nodenya. Tabel 3.1 menunjukan spesifikasi cluster berdasarkan arsitektur pada Gambar 3.4. Pemilihan 2 vCPU pada setiap node memberikan kemampuan pemrosesan yang cukup untuk menangani tugas-tugas pengelolaan cluster dan aplikasi yang dijalankan di atasnya. Hal ini sejalan dengan tujuan skripsi untuk melakukan deployment layanan dan komponen yang diperlukan tanpa menghadapi bottleneck kinerja yang signifikan. Konfigurasi 2 vCPU pada master dan worker nodes memungkinkan distribusi beban kerja yang lebih merata dan memastikan bahwa operasi Kubernetes seperti scheduling, controlling, dan monitoring dapat berjalan dengan lancar.

Memilih 4 GB vRAM untuk setiap node memberikan ruang memori yang cukup untuk menjalankan berbagai pods dan containers yang diperlukan untuk eksperimen. Ini penting untuk memastikan bahwa aplikasi dapat berfungsi dengan baik tanpa mengalami masalah memori. Dengan 4 GB RAM, node master dapat mengelola state cluster dengan efisien, sementara worker nodes dapat menjalankan aplikasi dan layanan dengan stabil. Ini membantu mencapai tujuan skripsi untuk melakukan deployment yang efisien dan stabil.

Kapasitas penyimpanan 30 GB per node menyediakan ruang yang memadai untuk menyimpan data aplikasi, logs, dan konfigurasi yang diperlukan selama eksperimen. Ini juga mencakup ruang untuk images Docker yang dibutuhkan oleh pods yang di-deploy. Spesifikasi storage ini cukup untuk kebutuhan penelitian, di mana data yang dihasilkan dan diproses mungkin tidak terlalu besar, tetapi membutuhkan ruang yang cukup untuk kelancaran operasi

Tabel III.1 Spesifikasi

node	*CPU*	*RAM*	*Storage*
Master	4	4	30
Worker 1	4	4	30
Worker 2	4	4	30

C. Variable Evaluasi

- *Pod Count*

  Variabel ini merujuk pada jumlah *pod* yang dibuat atau di-deploy dalam lingkungan *Kubernetes* untuk menangani pengolahan data. *Pod* merupakan unit terkecil dari deployment dalam *Kubernetes* yang terdiri dari satu atau beberapa container yang saling terkait. Evaluasi terhadap *Pod Count* membantu dalam memahami skalabilitas *Horizontal* sistem, yakni seberapa baik sistem mampu menambah atau mengurangi jumlah *pod* secara dinamis berdasarkan beban kerja yang ada. Pengujian ini mempengaruhi performa keseluruhan

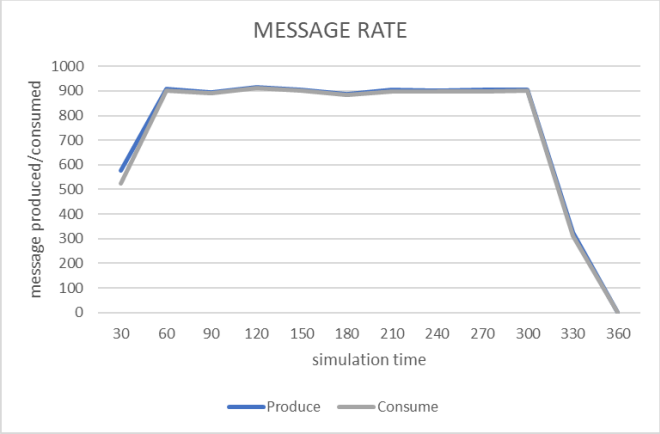
sistem dalam menangani tugas-tugas pemrosesan data pada IoT

- *Message Rate*
  Variabel ini mengacu pada tingkat atau kecepatan pesan atau data yang masuk ke sistem untuk diproses. Evaluasi terhadap message rate memberikan gambaran tentang seberapa cepat sistem mampu menangani arus data yang masuk dari perangkat IoT. Analisis pada variabel ini membantu dalam menentukan sejauh mana sistem mampu memproses pesan atau data dalam waktu yang ditentukan. Keefisienan dalam mengelola dan memproses Message Rate memengaruhi kinerja sistem dalam merespons peristiwa atau data dari berbagai sumber IoT

IV. HASIL DAN ANALISIS

A. Message Rate KEDA (*Average Based Metric*)

Berikut adalah grafik message rate dengan treshold autoscaling menggunakan average based metric:



Gambar IV.1 *Message Rate KEDA*

Pada periode waktu 60 hingga 300 produce dan consume rate tetap konsisten dengan rata-rata message rate sekitar 90% keatas, hal ini menunjukan sistem yang stabil dimana message atau data yang masuk dan diproses cenderung sama. Pada detik 330 terjadi penurunan tajam dalam tingkat pesan yang diproduksi dan dikonsumsi, hal ini disebabkan oleh fakta bahwa produser telah berhenti mengirim pesan setelah 300 detik. Oleh karena itu, yang kita lihat pada detik 330 adalah sisa dari pesan yang masih belum diproses dan dikonsumsi pada message broker kafka. Karena tidak ada pesan baru yang diproduksi, tingkat produksi turun menjadi nol atau mendekati nol, sementara tingkat konsumsi juga menurun karena consumer hampir menyelesaikan pemrosesan message yang tersisa.

B. Message Rate PCM *(Count Based Metric)*

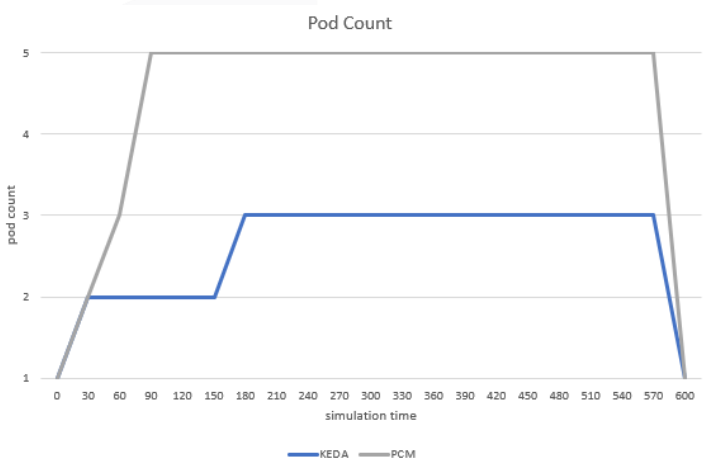
. Message rate pada metode count-based metric cenderung sama dengan average based metric dengan rata-rata message ratet pada setiap periode waktu simulasi mencapai diatas 90%. Hal ini menunjukkan bahwa kedua metode autoscaling memiliki performa yang baik dalam menangani message rate.

Gambar IV.2 Message Rate PCM

C. *Pod Count*

Pada penelitian ini, pengukuran jumlah pod dilakukan untuk memahami perilaku HPA pada setiap metode. Data jumlah pod ini kemudian dianalisis untuk menentukan korelasi antara metode yang digunakan dengan efisiensi hasil yang diperoleh, serta untuk mengevaluasi efektivitas masing-masing metode dalam meningkatkan produktivitas autoscaling.

Berikut merupakan grafik yang menunjukkan pod count atau jumlah pod yang scaling pada periode waktu tertentu:



Gambar IIV.3 *Pod Count*

Gambar IV.3 menunjukkan grafik grafik jumlah pod yang di scaling oleh masing-masing metode autoscaling. Pada detik 30 metode autoscaling KEDA dan PCM melakukan scaling pod dengan jumlah yang sama yaitu sebanyak 2 pod. Pada detik 60 hingga 150 KEDA masih mempertahankan jumlah pod sedangkan PCM melakukan scaling kembali pada detik 60 sebanyak 3 pod. Pada detik 90 PCM mencapai pod maksimal hingga detik 570 dengan pod sebanyak 5 sedangkan KEDA scaling dengan total pod sebanyak 3 dan mempertahankan secara konstan hingga pod melakukan scaling down.

Berdasarkan analisis, KEDA memiliki jumlah pod yang lebih sedikit dibandingkan PCM selama periode simulasi, hal ini membuat KEDA menunjukkan efisiensi yang lebih tinggi dalam mempertahankan performa dengan sumber daya yang lebih sedikit. Hal ini mengindikasikan bahwa metode

autoscaling KEDA lebih stabil dan efisien dibandingkan PCM, terutama dalam hal penggunaan sumber daya

V. KESIMPULAN

Berdasarkan tahapan dan hasil pengujian dalam penelitian ini, dapat disimpulkan bahwa kedua metode autoscaling sama-sama memiliki tingkat message rate yang stabil dan tinggi. Tetapi pada pengukuran jumlah pod, metode autoscaling menggunakan KEDA memiliki jumlah pod yang lebih sedikit hampir 50% dibandingkan PCM. Hal ini dikarenakan KEDA melakukan scaling pod berdasarkan rata-rata jumlah consumer lag setiap periode interval sehingga ambang batas autoscaling dan jumlah pod tetap stabil.

Dalam penelitian ini juga ditemukan bahwa KEDA lebih efisien dalam penggunaan sumber daya karena dapat mengurangi jumlah pod yang berjalan tanpa mengorbankan kinerja message rate dengan rata-rate message yang di konsumsi mencapai 90 – 100%. Sebaliknya, PCM cenderung menghasilkan jumlah pod yang lebih besar karena mekanisme scaling-nya yang tidak seefisien KEDA dalam menyesuaikan dengan beban kerja. Oleh karena itu, KEDA dapat dianggap sebagai metode autoscaling yang lebih unggul dalam hal efisiensi sumber daya dan stabilitas jumlah pod, terutama dalam konteks sistem berbasis IoT yang memerlukan penanganan beban kerja yang bervariasi secara dinamis

REFERENSI

[1] "Digital Around the World — DataReportal – Global Digital Insights." Accessed: Oct. 29, 2023. [Online]. Available: https://datareportal.com/global-digital-overview

[2] J. Surbiryala and C. Rong, "*Cloud* computing: History and overview," in *Proceedings - 2019 3rd IEEE International Conference on Cloud and Fog Computing Technologies and Applications, Cloud Summit 2019*, Institute of Electrical and Electronics Engineers Inc., Aug. 2019, pp. 1–7. doi: 10.1109/*Cloud*Summit47114.2019.00007.

[3] L. Minh Dang, M. J. Piran, D. Han, K. Min, and H. Moon, "A survey on internet of things and *cloud* computing for healthcare," *Electronics (Switzerland)*, vol. 8, no. 7, Jul. 2019, doi: 10.3390/electronics8070768.

[4] R. Sikarwar, P. Yadav, and A. Dubey, "9 th IEEE International Conference on Communication Systems and Network Technologies A Survey on IOT enabled *cloud* platforms," *2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT)*, 2020, doi: 10.1109/CSNT.2020.23.

[5] G. Paolone, D. Iachetti, R. Paesani, F. Pilotti, M. Marinelli, and P. Di Felice, "A Holistic Overview of the Internet of Things Ecosystem," *Internet of Things*, vol. 3, no. 4. MDPI, pp. 398–434, Dec. 01, 2022. doi: 10.3390/iot3040022.

[6] H. Yang and Y. Kim, "Design and implementation of high-availability architecture for IoT-*Cloud* services," *Sensors (Switzerland)*, vol. 19, no. 15, Aug. 2019, doi: 10.3390/s19153276.

[7] R. Berjón, M. Mateos, M. E. Beato, and A. F. García, "An *Event* Mesh for *Event Driven* IoT Applications," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 7, no. 6, pp. 54–59, 2022, doi: 10.9781/ijimai.2022.09.003.

[8] V. Ana Maria Zambrano, V. Marcelo Zambrano, E. L. O. Mejia, and H. Xavier Calderon, "SIGPRO: A Real-Time Progressive Notification System Using MQTT Bridges and Topic Hierarchy for Rapid Location of Missing Persons," *IEEE Access*, vol. 8, pp. 149190–149198, 2020, doi: 10.1109/ACCESS.2020.3015183.

[9] "IoT notification system for marine emergencies", doi: 10.7236/IJIBC.2022.14.1.122.

[10] S. S. Mazlan, N. Mohamed, and F. Majid, "Development of Monitoring System using Raspberry Pi with Instant Notification," MBOT, 2023.

[11] M. Umar Diginsa *et al.*, "Low-cost IoT-Based Smart Notification System for Rural Agriculture Article history," 2023.

[12] S. Misbahuddin, M. Mohsen, M. Abdulqudus, A. Ahmed, and A. K. Bin Jahlan, "IoT based Automatic Email and Audio *Message* Notification System for Electric Light Failure Detection inside Harmain Sharifain of Makkah and Madinah," in *2020 International Conference on Computing and Information Technology, ICCIT 2020*, Institute of Electrical and Electronics Engineers Inc., Sep. 2020. doi: 10.1109/ICCIT-144147971.2020.9213811.

[13] D. Angelo, B. Go Carl, J. M. Deiparine, J. C. Cuizon, and B. L. Canonigo, "Presence: An Integrated Mobile Solution for Truancy Detection using RFID and *Cloud*-based Notification Services," 2018.

[14] I. C. Donca, C. Corches, O. Stan, and L. Miclea, "Autoscaled RabbitMQ *Kubernetes Cluster* on single-board computers," in *2020 22nd IEEE International Conference on Automation, Quality and Testing, Robotics - THETA, AQTR 2020 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., May 2020. doi: 10.1109/AQTR49680.2020.9129886.

[15] T. T. Nguyen, Y. J. Yeom, T. Kim, D. H. Park, and S. Kim, "*Horizontal pod autoscaling* in *Kubernetes* for elastic container orchestration," *Sensors (Switzerland)*, vol. 20, no. 16, pp. 1–18, Aug. 2020, doi: 10.3390/s20164621.

[16] P. Chindanonda, V. *Pod*olskiy, and M. Gerndt, "*Metric*s for self-adaptive queuing in middleware for internet of things," in *Proceedings - 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems, FAS*W 2019*, Institute of Electrical and Electronics Engineers Inc., Jun. 2019, pp. 130–133. doi: 10.1109/FAS-W.2019.00042.

[17] P. Chindanonda, V. *Pod*olskiy, and M. Gerndt, "Self-adaptive data processing to improve SLOs for dynamic iot workloads," *Computers*, vol. 9, no. 1, Mar. 2020, doi: 10.3390/computers9010012.

[18] M. Chadha, V. Pacyna, A. Jindal, J. Gu, and M. Gerndt, "Migrating from Microservices to Serverless: An IoT Platform Case Study," in *WoSC 2022 - Proceedings of the 8th International Workshop on Serverless Computing, Part of Middleware 2022*, Association for Computing Machinery, Inc, Nov. 2022, pp. 19–24. doi: 10.1145/3565382.3565881.

[19] M. Mouine and M. A. Saied, "*Event-Driven* Approach for Monitoring and Orchestration of *Cloud* and Edge-Enabled IoT Systems," in *IEEE International Conference on Cloud Computing, CLOUD*, IEEE Computer Society, 2022, pp. 273–282. doi: 10.1109/*CLOUD*55607.2022.00049.

[20] S. Qabil, U. Waheed, S. M. Awan, Y. Mansoor, and M. A. Khan, "A Survey on Emerging Integration of *Cloud* Computing and Internet of Things."

[21] D. Bastos, "*Cloud* for IoT - A survey of technologies and security features of public *cloud* IoT solutions," in *IET Conference Publications*, Institution of Engineering and Technology, 2019. doi: 10.1049/cp.2019.0168.

[22] "*Kubernetes* Documentation | *Kubernetes*." Accessed: Dec. 07, 2023. [Online]. Available: https://*Kubernetes*.io/docs/home/

[23] "The *KEDA* Documentation | *KEDA*." Accessed: Dec. 07, 2023. [Online]. Available: https://*KEDA*.sh/docs/2.12/