Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 4, No. 1, Januari 2020, hlm. 100-108 http://j-ptiik.ub.ac.id

Pengembangan Infrastruktur Analisis Data *Heart Rate* berbasis *Microservices* menggunakan Kubernetes

Abd. Jahiduddin[1], Eko Sakti Pramukantoro[2], Fariz Andri Bakhtiar[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]abd.jahiduddin@student.ub.ac.id, [2]ekosakti@ub.ac.id, [3]fariz@ub.ac.id

Abstrak

Peningkatan jumlah perangkat *Internet of Things* setiap harinya menyebabkan peningkatan volume data yang tidak memiliki manfaat jika tidak dilakukan pemrosesan data. Tantangan paling signifikan dalam melakukan pemrosesan data adalah kebutuhan sumber daya, proses instalasi yang kompleks, kesulitan dalam hal penggunaan alat analisis, serta proses pemeliharaan dan pengembangan yang rumit. Penelitian ini mencoba untuk menyelesaikan masalah tersebut dengan mengembangkan infrastruktur analisis data *heart rate* menggunakan arsitektur *microservices*, Docker *container* dan Kubernetes. Penggunaan arsitektur *microservices* bertujuan untuk mempermudah pengembangan dan pemeliharaan aplikasi. Penggunaan docker untuk mempermudah proses instalasi dan *deployment* dengan mengubah aplikasi dengan arsitektur microservices menjadi docker *image*. Penggunaan Kubernetes untuk melakukan manajemen terhadap docker *container* yang berjalan pada cluster Kubernetes. Pengujian skalabilitas menunjukkan infrastruktur yang dibangun mampu melakukan analisis hingga 100.000 data. Namun, saat melakukan analisis 50.000 data terjadi peningkatan waktu analisis secara drastis sebanyak tujuh hingga delapan kali lipat. Selain kemampuan analisis, penelitian ini juga menghasilkan infrastruktur analisis data *heart rate* yang memiliki kemampuan dalam melakukan *self-healing* dan mampu menyimpan data secara *persistent*.

Kata kunci: *cloud computing*, infrastruktur data analisis, *microservices*, Docker, Kubernetes, *container*, *container orchestration*

Abstract

Increasing the number of Internet of Things devices every day causes an increase in the volume of data that has no benefit if not done data processing. The most significant challenges in data processing are resource requirements, complex installation processes, difficulties in using analytical tools, and complex maintenance and development processes. This research tries to solve this problem by developing a heart rate data analysis infrastructure using microservices architecture, Docker containers, and Kubernetes. The use of microservices architecture aims to facilitate the development and maintenance of applications. The use of a docker to simplify the installation and deployment process by changing applications with microservices architecture to a docker image. Kubernetes is used to carry out the management of the container docker that runs on the Kubernetes cluster. Scalability testing shows that the infrastructure built is capable of analyzing up to 100,000 data. However, when analyzing 50,000 data there was a drastic increase in an analysis time of seven to eight times. In addition to analytical skills, this study also produces a heart rate data analysis infrastructure that can perform self-healing and can store data persistently.

Keywords: *cloud computing, infrastructure data analytic, microservices, docker, kubernetes, container, container orchestration*

1. PENDAHULUAN

Setiap harinya semakin banyak jumlah perangkat *Internet of Things* (IoT) yang terhubung di seluruh dunia (Bashir & Gill, 2016). *Cisco Internet Business Solutions Group* (IBSG) memprediksi di tahun 2020 akan ada 50 milyar perangkat yang terhubung ke internet dan pada tahun 2010 jumlah perangkat yang terhubung ke internet telah melebihi jumlah populasi dunia (Evans, 2011).

Peningkatan jumlah perangkat *Internet of Things* menyebabkan peningkatan data dalam jumlah yang sangat besar, data tersebut tidak akan berguna jika tidak dilakukan pemrosesan data (Marjani, et al., 2017)(Al-Fuqaha, et al., 2015). Tantangan paling signifikan adalah kebutuhan sumber daya dalam pemrosesan data, proses instalasi yang kompleks, kesulitan dalam hal penggunaan alat untuk analisis, serta proses pemeliharaan dan pengembangan yang rumit (Naik, 2017).

Pada penelitian yang dilakukan oleh Agus Adyandana yang berjudul "Pengembangan *Lightweight* Data *Analytic* Pada Perangkat *Edge*", telah dikembangkan aplikasi analisis data *heart rate* yang ringan (*lightweight* data *analytic*) pada dalam lingkungan *edge computing* menggunakan perangkat Raspberry Pi, dengan tujuan untuk mendapatkan informasi mengenai kondisi detak jantung (Adyandana, Pramukantoro, & Bakhtiar, 2019).

Pada penelitian lain yang dilakukan oleh Xu dkk yang berjudul "*Performance Evaluation of Deep Learning Tools in Docker Containers*". Penelitian tersebut dilakukan untuk mengetahui dampak penggunaan docker *container* terhadap kinerja model *deep learning*. Hasil pengujian menunjukkan aplikasi yang berjalan di atas docker *container* sama baiknya dengan aplikasi yang berjalan di atas *host system* (Xu, Shi, & Chu, 2017).

Penelitian yang dilakukan oleh Adyandana dkk telah menjawab permasalahan mengenai pemrosesan data. Keterbatasan sumber daya yang dimiliki oleh Perangkat Raspberry Pi menyebabkan waktu pemrosesan data menjadi lebih lama ketika volume data yang masuk ke perangkat Raspberry Pi meningkat. Sedangkan pada penelitian Xu dkk telah menjawab masalah mengenai proses instalasi yang kompleks dan kesulitan dalam hal penggunaan alat untuk analisis. Kemampuan isolasi dari docker mempermudah proses *deployment* dan menjalankan berbagai aplikasi dengan cepat di dalam sebuah *container* (Naik, 2017). Dikarenakan penelitian Xu dkk tidak menyediakan *container orchestration*, secara praktis manajemen *container* memberi kompleksitas tambahan (Shah & Dubaria, 2019) pada implementasi sistem secara umum.

Terdapat berbagai jenis container orchestration yang dapat digunakan yaitu docker swarm dan Kubernetes. Marathe dkk dalam penelitiannya menggunakan docker swarm dan Kubernetes menemukan kinerja dari CPU dan *memory* pada Kubernetes lebih baik dari docker swarm. Kubernetes juga memiliki kemampuan *self-healing* yang tidak dimiliki oleh docker swarm (Marathe, Gandhi, & Shah, 2019).

Fokus penelitian ini adalah membangun infrastruktur analisis *heart rate* pada lingkungan *cloud computing* untuk mengatasi keterbatasan sumber daya pada lingkungan *edge analytic*, karena *cloud computing* memiliki *resource* yang besar dan cenderung tak terbatas untuk memproses data (Faathin, Pramukantoro, & Bakhtiar, 2019). Docker digunakan agar proses instalasi dan penggunaan alat analisis lebih mudah. Untuk mempermudah proses pemeliharaan dan pengembangan aplikasi yang berjalan di atas docker *container*, digunakan arsitektur *microservices* (Kang, Le, & Tao, 2016). Kubernetes digunakan untuk melakukan manajemen container karena kinerja yang lebih baik dalam hal penggunaan CPU dan memory, kemampuan manajemen *cluster*, serta fitur *network* yang dimiliki.

2. LANDASAN KEPUSTAKAAN

Pada penelitian yang dilakukan oleh Agus Adyandana telah dikembangkan aplikasi analisis data yang ringan (*lightweight* data *analytic*) dalam lingkungan *edge computing* pada perangkat Raspberry Pi Zero. Data yang digunakan pada penelitian tersebut adalah data heart rate. Data nantinya dianalisis dengan cara melakukan klasifikasi data menggunakan algoritma *Decision Tree* dan *tools* Scikit-Learn (*Machine Learning*). Data hasil analisis diklasifikasikan menjadi tiga yaitu normal, *fast*, dan *slow* (Adyandana, Pramukantoro, & Bakhtiar, 2019)

Pada penelitian lain yang dilakukan oleh Xu dkk. Pada penelitian tersebut mencoba menjalankan beberapa *tools deep learning* di atas *container*. Hasil dari penelitian tersebut menunjukkan aplikasi yang berjalan di atas docker *container* sama baiknya dengan *applikasi* yang berjalan di atas *host system*. Sehingga menjalankan aplikasi *deep learning* di dalam docker *container* merupakan solusi yang layak serta mendapatkan fleksibilitas, *lightweight* dan kemampuan dalam mengisolasi aplikasi oleh docker *container* (Xu, Shi, & Chu, 2017)

2.1. Cloud Computing

Cloud computing di perkenalkan sebagai sebuah pendekatan baru dari pemrosesan data di mana sumber daya tersedia secara virtual

sebagai sebuah *service* melalui internet. Pada zaman sekarang ini *cloud computing* menjadi sangat penting untuk pengembangan dan para ahli di berbagai bidang mengharapkan bahwa *cloud computing* akan mengubah pemrosesan data untuk kebutuhan riset dan bisnis dalam industri IT (Pawar & Attar, 2016).

2.2. Microservices

Microservices memiliki kemampuan dalam mengurangi kerumitan dalam pengembangan dan penyebaran aplikasi karena membagi aplikasi besar menjadi layanan yang lebih kecil (Amaral, et al., 2016). Dengan menggunakan *microservices* dapat dilakukan *deployment*, *scaling* dan pengujian secara terpisah. (Thönes, 2015). Setiap *microservices* memiliki prosesnya sendiri dan dapat saling berkomunikasi menggunakan mekanisme komunikasi seperti RESTful atau API berbasis RPC (Balalaie, Heydarnoori, & Jamshidi, 2016)

2.3. Docker

Docker menggunakan arsitektur *client-server*. Docker *client* berkomunikasi dengan docker *daemon* menggunakan REST API melalui UNIX *socket* atau *network interface*. Docker memiliki tiga komponen docker yaitu *client*, *daemon*, dan *registry* yang digunakan pada penelitian ini.

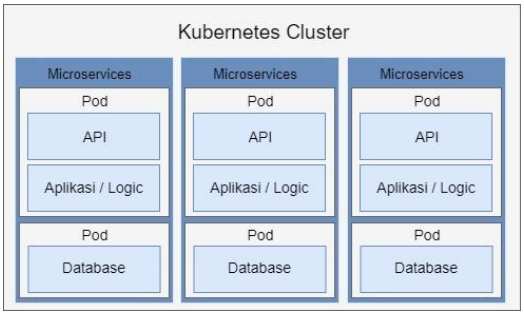
Docker *daemon* berjalan di host berfungsi untuk melakukan manajemen terhadap docker *object* seperti *image* dan *container*. Docker *client* merupakan media agar pengguna dapat berinteraksi dengan docker. Docker *registry* merupakan tempat penyimpanan docker *image* dan *base image* yang biasanya digunakan membuat *image* lain. Salah satu contoh docker *regsitry* adalah docker hub yang dapat digunakan oleh siapa saja (Docker, t.thn.).

2.4. Kubernetes

Kubernetes merupakan *container orchestration* atau *opensource* platform untuk melakukan manajemen terhadap *containerized services* (Smarvin, 2019). Pada penelitian ini digunakan beberapa *object* kubernetes yaitu.

- *Pod* merupakan unit dasar dari Kubernetes. *Pod* mewakili proses yang berjalan pada suatu *cluster* kubernetes (Vageesha17, 2019a).
- *Deployment* bertugas untuk melakukan instruksi kepada Kubernetes tentang bagaimana suatu *pod* dibuat dan diperbarui (tddorgtfo, 2019).
- Service merupakan suatu cara untuk mengekspos aplikasi yang berjalan di *pod* sebagai suatu layanan jaringan (Jimangel, 2019).
- *Statefulset* hampir sama dengan *deployment* berfungsi untuk melakukan *deploy docker image* ke sebuah *pod* dan memastikan *pod* tetap berjalan. Perbedaannya *statefulset* digunakan untuk melakukan manajemen terhadap aplikasi yang bersifat *statefull*. (Vageesha17, 2019b)

3. ANALISIS KEBUTUHAN



Gambar 1. Cluster Kubernetes

Sistem ini dikembangkan dengan tujuan untuk menyediakan sebuah infrastruktur yang dapat melakukan analisis data *heart rate*, menggunakan arsitektur *microservices*, berbasis docker *container* dan Kubernetes untuk melakukan manajemen terhadap *container* pada lingkungan *cloud*. Aplikasi *heart rate analytic* yang telah dikembangkan pada penelitian sebelumnya (Adyandana, et al., 2019) dipecah menjadi beberapa unit fungsional yang lebih kecil. Setiap unit fungsional diubah menjadi docker *image*. Setelah itu, dilakukan proses *deployment* ke *pod* yang terdapat pada *cluster* kubernetes.

3.1. Kebutuhan Microservices

Infrastruktur yang dibangun harus memiliki kemampuan menerima data dari sensor, analisis data, dan visualisasi data. Sehingga dibangun tiga *microservices* yaitu *user dashboard*, *heart rate analytic*, dan *gateway*.

Microservices user dashboard sebagai *user interface* untuk mengakses keseluruhan sistem seperti melakukan request analisis data ke *microservices heart rate analytic* dan melakukan visualisasi data.

Microservices heart rate analytic berfungsi

untuk melakukan analisis data *heart rate*. Data yang dianalisis terdapat di *microservices gateway*, sehingga dilakukan *request* data terlebih dahulu sebelum melakukan analisis data.

Microservices gateway berfungsi untuk menerima data dari *dummy* sensor melalui protokol HTTP. Sensor yang dapat mengirimkan data ke *microservices* ini hanya sensor yang telah terdaftar pada sistem.

Setiap *microservices* harus memiliki kemampuan untuk melakukan proses komputasi, menyimpan data, dan berkomunikasi dengan *microservices* lain. Sehingga setiap *microservices* memiliki tiga komponen utama yaitu *database*, API, dan aplikasi. *Database* digunakan sebagai media penyimpanan data. API digunakan sebagai media komunikasi antar *microservices*. Aplikasi digunakan untuk melakukan proses komputasi atau menjalankan logika tertentu.

3.2. Kebutuhan Object Kubernetes

Kubernetes memiliki beberapa *object* yang digunakan pada penelitian ini yaitu *pod, deployment, service, statefulset, persistentvolume*, dan *persistemvolumeclaim*. *Pod* digunakan untuk menjalankan aplikasi *microservices*. Namun, *pod* memiliki siklus hidup yang dapat menyebabkan *pod* mati sehingga aplikasi yang berjalan di dalam *pod* tidak dapat diakses serta data yang tersimpan di dalam *pod* juga akan ikut terhapus. Sehingga untuk mengatasi masalah tersebut digunakan *object deployment* dan *statefulset*.

Deployment dan *statefulset* digunakan untuk menjalankan *pod* dan memastikan *pod* selalu dalam kondisi berjalan. *Deployment* digunakan untuk aplikasi yang bersifat stateless sedangkan *statefulset* digunakan untuk aplikasi yang bersifat *statefull*. Namun, untuk menggunakan *statefulset* perlu dilakukan pembuatan *persistenvolume* dan *persistentvolumeclaim*. *Persistentvolume* berfungsi untuk menyediakan sumber daya penyimpanan. Sedangkan *persistentvolumeclaim* digunakan untuk melakukan klaim terhadap sumber daya yang telah disediakan oleh *persistentvolume*.

Semua *pod* harus dapat diakses dari dalam atau luar *cluster* kubernetes sehingga digunakan *object service* dengan tipe ClusterIP dan NodePort untuk mengekspos *pod* sebagai suatu layanan jaringan. Service tipe ClusterIP memungkinkan *pod* diakses dari di dalam *cluster* kubernetes, sedangkan NodePort memungkinkan *pod* diakses dari luar cluster kubernetes.

3.3 Lingkungan Penelitian

Lingkungan penelitian yang ada pada penelitian ini terdiri dari lingkungan pengembangan, *deployment* dan pengujian. Setiap lingkungan tersebut memiliki kebutuhan perangkat keras dan lunak yang berbeda-beda. Kebutuhan perangkat keras dapat dilihat pada tabel 1 dan kebutuhan perangkat lunak dapat dilihat pada tabel 2.

Tabel 1. Kebutuhan Perangkat Keras

Perangkat Keras	Keterangan
VPS	VPS digunakan untuk menjalankan *cluster* Kubernetes secara terdistribusi. Untuk menjalankan *cluster* kubernetes dibutuhkan minimal *memory* 2 GB dan 2 CPUs.

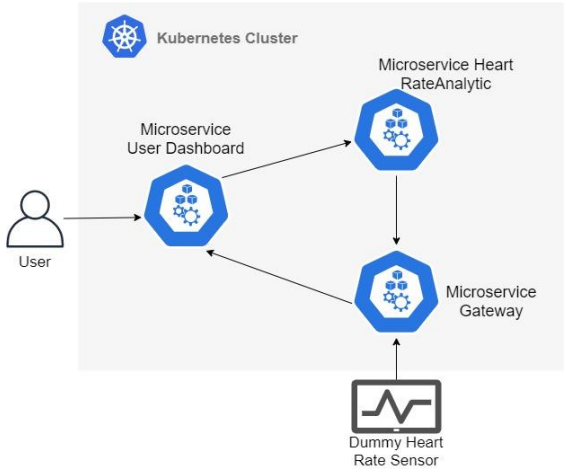
Tabel 2. Kebutuhan Perangkat Lunak

Perangkat Lunak	Keterangan
Sistem Operasi	Sistem operasi yang digunakan berbasis linux dengan distro ubuntu desktop & server.
Visual Studio Code	*Text editor* yang digunakan untuk melakukan pengembangan aplikasi
Django	Django digunakan dalam membangun komponen aplikasi dan API.
MongoDB	MongoDB digunakan sebagai *database* utama di setiap service untuk melakukan penyimpanan data.
Docker engine	Docker digunakan untuk membuat docker *image*.
Kubernetes	Kubernetes akan digunakan untuk melakukan manajemen *container*.

4. PERANCANGAN SISTEM

4.1. Perancangan Arsitektur

Secara keseluruhan arsitektur sistem yang dikembangkan dapat dilihat pada gambar 2 :



Gambar 2. Arsitektur Sistem

Keterangan dari gambar 2:

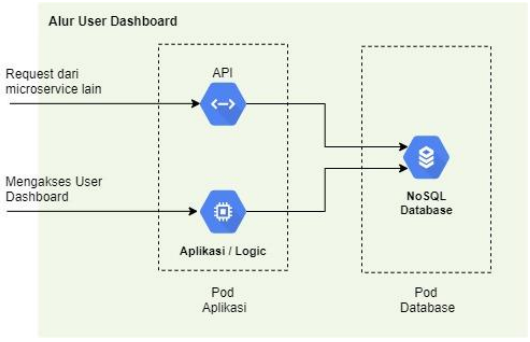
1. Terdapat tiga *microservices* yaitu user *dashboard*, *heart rate analytic*, dan *gateway*. Setiap microservices dideploy ke pod yang berbeda.
2. Pengguna mengakses microservice user dashboard untuk mengakses keseluruhan sistem melalui web.
3. Sensor mengirimkan data ke *microservice gateway*.
4. Setiap *microservices* dapat saling berkomunikasi melalui API, untuk penjelasan lebih lengkap mengenai alur komunikasi dapat dilihat pada bagian 4.2 Perancangan Alur Sistem

4.2 Perancangan Alur Sistem

Alur sistem pada penelitian ini dibagi menjadi dua bagian yaitu alur sistem antar *microservices* dan alur sistem antara komponen yang terdapat pada setiap *microservices*. Sehingga memberikan gambaran bagaimana sistem bekerja pada level *cluster* Kubernetes dan pada level *microservices*.

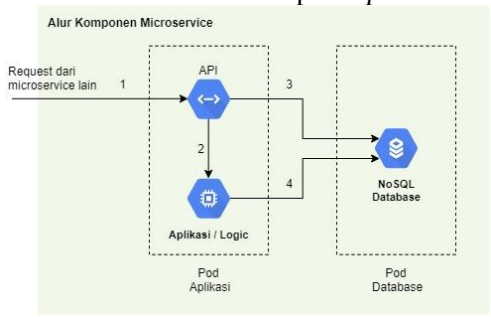
4.2.1 Alur Sistem Komponen Microservices

Alur sistem pada komponen *microservice heart rate analytic* dan gateway adalah sama sehingga dijelaskan ke dalam satu bagian, sedangkan alur sistem pada komponen *microservice user dashboard* berbeda dengan komponen *microservices* lainnya. Alur komunikasi pada komponen *microservices* dapat dilihat gambar 4 dan 5.



Gambar 4. Alur Sistem Komponen *Microservices User Dashboard*

Pada gambar 4 dapat dilihat pengguna dapat mengakses langsung komponen aplikasi melalui *user interface* berupa aplikasi web, hal ini yang membedakan alur komunikasi komponen *microservices user dashboard* dan lainnya. Alur komunikasi dibagi menjadi dua. Pertama, pengguna mengakses komponen aplikasi melalui web untuk menjalankan fungsionalitas yang dimiliki oleh *microservices user dashboard*. Komponen aplikasi mengakses *database* untuk mendapatkan informasi yang dibutuhkan. Alur komunikasi kedua, *microservices* lain mengakses API dari *microservice user dashboard* kemudian API melakukan *query* ke *database* untuk mendapatkan data yang dibutuhkan kemudian dikembalikan oleh API berupa *response*.



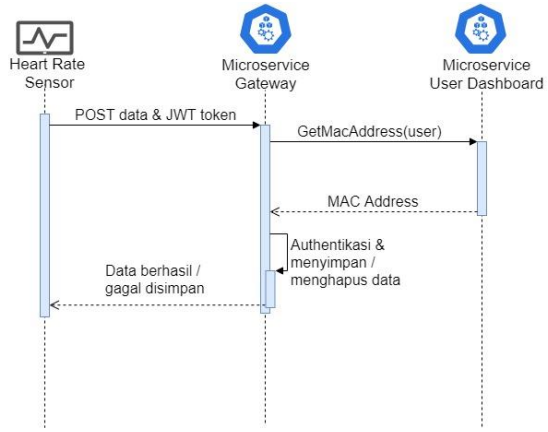
Gambar 5. Alur Sistem Komponen Microservices HRA & Gateway

Komponen *microservices heart rate analytic* dan *gateway* memiliki alur sistem yang sama. Pada gambar 5 diberikan penomoran untuk mempermudah peneliti menjelaskan alur sistem. Pertama alur sistem dengan urutan 1-2-4, *request* masuk ke komponen API kemudian menjalankan proses komputasi yang terdapat pada komponen aplikasi, kemudian komponen aplikasi melakukan *query* ke *database* untuk mendapatkan data yang diperlukan untuk melakukan proses komputasi atau untuk menyimpan hasil komputasi. Alur komunikasi

yang kedua 1-3, *request* masuk ke komponen API kemudian melakukan *query* ke *database* dan data hasil *query* dikembalikan dalam bentuk *response*.

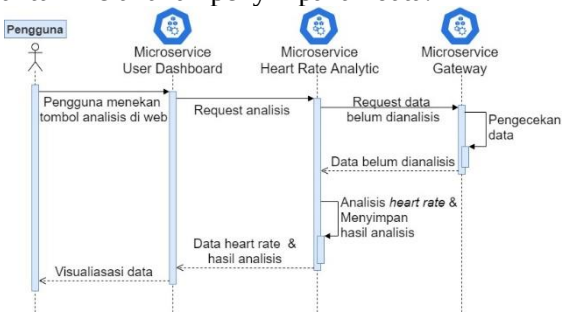
4.2.2 Alur Sistem Microservices

Perancangan alur sistem *microservices* merupakan rancangan alur komunikasi antara *microservices* yang dikembangkan pada penelitian ini. Secara umum ada tiga alur komunikasi utama yang dibangun pada penelitian ini yaitu alur pengiriman data ke *gateway*, alur analisis data dan alur visualisasi data.



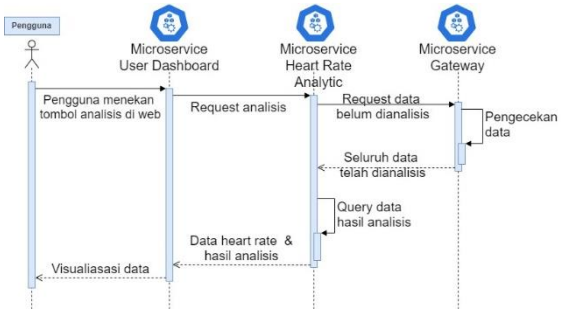
Gambar 6. Sequence Diagram Alur Pengiriman Data Ke Gateway

Alur komunikasi pengiriman data ke *gateway* dapat dilihat pada gambar 6. Pada alur komunikasi pengiriman data terdapat mekanisme *authentikasi* untuk memastikan *device* yang mengirim data mempunyai hak untuk melakukan penyimpanan data.



Gambar 7. Sequence Diagram Alur Analisis Data

Alur analisis data dapat dilihat pada gambar 7. Pada alur komunikasi analisis data dilakukan pengecekan dan query data yang belum dianalisis pada *microservices gateway*. Setelah itu dilakukan analisis data *heart rate* pada *microservice heart rate analytic*. Setelah analisis selesai dilakukan penyimpanan dan mengembalikan hasil analisis ke *microservice user dashboard*.



Gambar 8. Sequence Diagram Alur Visualisasi Data

Alur visualisasi data dapat dilihat pada gambar 8. Pada alur komunikasi visualisasi data dilakukan pengecekan data pada *microservice gateway*. Jika seluruh data telah dianalisis maka *microservices heart rate analytic* akan melakukan *query* data hasil analisis dan mengembalikan ke *microservice user dashboard*.

4.3 Perancangan Pengujian

Perancangan pengujian yang dirumuskan terdiri dari perancangan skalabilitas, *self-healing*, dan *persistent pod database*.

4.3.1 Perancangan Pengujian Skalabilitas

Pengujian ini bertujuan untuk mengetahui kemampuan dan batasan dari infrastruktur yang dibangun dalam melakukan analisis data. Pengujian skalabilitas ini menggunakan 5 skenario dengan pola berdasarkan penelitian Faathin dkk (Faathin, Pramukantoro, & Bakhtiar, 2019). Pola yang digunakan untuk melakukan pengujian adalah analisis 1.000, 5.000, 10.000, 50.000 dan 100.000 data. Pengujian dilakukan dengan cara melakukan *request* analisis menggunakan Postman.

4.3.2 Perancangan Pengujian Self-Healing

Pengujian bertujuan untuk mengetahui apakah *cluster* selalu dapat kembali ke keadaan normal jika terdapat *pod* yang mati. Penelitian ini dilakukan dengan cara menghapus salah satu *pod* yang sedang berjalan. Kemudian peneliti mengamati sistem untuk mengetahui apakah *cluster* mampu kembali ke keadaan normal setelah *pod* dihapus.

4.3.3 Perancangan Pengujian Persistent Pod Database

Pengujian ini bertujuan untuk mengetahui

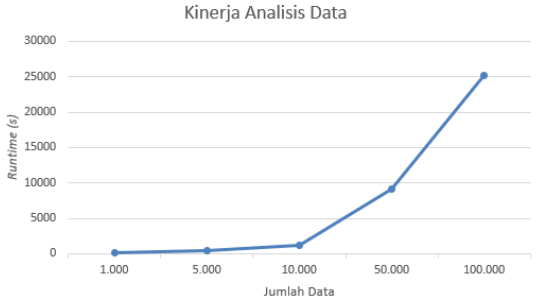
apakah data yang telah disimpan pada *database* tidak terhapus meskipun *pod database* mati. Pengujian ini dilakukan karena *pod* memiliki siklus hidup yang menyebabkan *pod* dapat mati dan *pod* yang telah mati tidak dapat dijalankan lagi, sehingga penyimpanan dan data yang ada di dalam *pod* juga terhapus. Pengujian dilakukan dengan cara menghapus *pod* dan dilakukan pengamatan data yang tersimpan pada *pod database* sebelum dan setelah *pod* dihapus. Pengujian ini dilakukan pada *pod database* setiap *microservices*.

5. HASIL DAN PEMBAHASAN

Pada bagian ini dijelaskan hasil pengujian yang telah dilakukan.

5.1 Hasil Pengujian Skalabilitas

Pengujian skalablitas dilakukan untuk mengetahui batasan infrastruktur yang telah dibangun dalam melakukan analisis data. Pengujian dilakukan menggunakan *tool* Postman untuk melakukan *request* analisis ke *microservice heart rate analytic*. Skenario uji yang dilakukan adalah dengan melakukan analisis 1.000, 5.000, 10.0000, 50.000, dan 100.000 data.

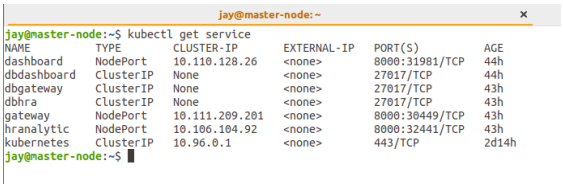


Gambar 9. Grafik Kinerja Analisis Data

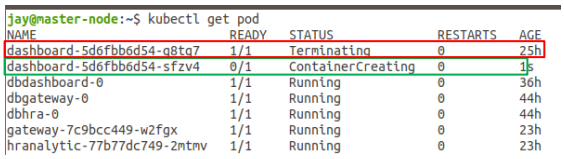
Pengujian analisis 1.000, 5.000, 10.000, 50.000 data berhasil melakukan analisis dan mengembalikan respon berupa waktu analisis, status, hasil analisis, dan total data yang telah di analisis. Pada gambar 7 hasil pengujian skenario 50.000 menunjukkan terjadi peningkatan waktu analisis secara drastis sebanyak 7 hingga 8 kali lipat dari pengujian analisis 10.000 data. Sedangkan, pengujian analisis 100.000 data berhasil melakukan keseluruhan proses analisis dengan waktu 2 hingga 3 kali lipat lebih lama dari pengujian 10.000 data. Namun, pada pengujian analisis 100.000 tidak dapat mengembalikan hasil analisis ke *client* yang melakukan *request* analisis.

5.2 Hasil Pengujian Self-Healing

Pengujian dilakukan dengan cara menghapus salah satu *pod*, kemudian dilakukan pengamatan, seluruh proses yang terjadi dimulai pada saat sebelum *pod* dihapus hingga *pod* baru berhasil berjalan untuk menggantikan *pod* yang mati.

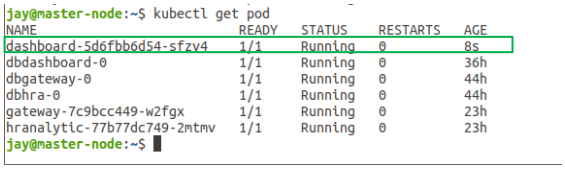


Gambar 10. Sebelum pod di hapus



Gambar 11. Pod lama dalam keadaan terminating

Gambar 12. Pod baru berjalan



Gambar 13 Cluster kembali ke keadaan normal

Pada gambar 10 seluruh status *pod* dalam keadaan berjalan dan jumlah *pod* yang berjalan sebanyak satu, kemudian dilakukan penghapusan *pod* dashboard-4dfbb6d54-q8tq7.

Pada gambar 11 setelah menghapus *pod* dashboard-4dfbb6d54-q8tq7 status *pod* tersebut berubah menjadi *Terminating*. *Deployment* menjalankan sebuah *pod* baru untuk menggantikan *pod* yang lama.

Pada gambar 12 jumlah *pod* baru yang siap telah bertambah menjadi satu dan status *pod* baru telah berubah menjadi *Running*, ini menandakan bahwa *pod* baru telah siap digunakan. Kemudian status pod lama masih dalam keadaan *Terminating* namun jumlah *pod* yang berjalan telah berubah menjadi 0.

Hasil pengujian *self-healing* yang telah dilakukan menunjukkan *cluster* dapat kembali

ke keadaan normal setelah dilakukan penghapusan *pod*, pada gambar 13 menunjukkan *pod* lama telah dihapus dan *cluster* kembali ke keadaan normal. Pengujian ini juga menunjukkan bahwa *pod* akan selalu tersedia untuk diakses hal ini dapat dilihat proses *self-healing* yang terjadi. *Deployment* membuat *pod* baru terlebih dahulu dan menunggu hingga *pod* baru dalam keadaan berjalan, kemudian *pod* lama benar-benar dihapus dari *cluster*.

5.3. Hasil Pengujian Persistent Pod Database

Pod database dikatakan *persistent* jika data yang tersimpan pada *database* tidak terhapus meskipun *pod database* mati dan *cluster* menjalankan *pod database* baru. Pengujian ini dilakukan pada setiap *pod database* yang dimiliki masing-masing *microservices*. Pengujian dilakukan dengan cara melakukan perbandingan data yang tersimpan pada *database* setelah dan sebelum *pod* dihapus.

Tabel 3. Hasil Pengujian Persistent Pod Database

Jenis Pod Database	Persistent atau Nonpersistent
User Dashboard	*Persisten*
Heart Rate Analytic	*Persisten*
Gateway	*Persisten*

Hasil pengujian pada tabel 3 menunjukkan seluruh data pada *pod database* tidak terhapus meskipun terjadi kondisi yang menyebabkan *pod database* mati dan *deployment* menjalankan *pod* baru untuk menggantikan *pod* yang mati.

6. KESIMPULAN

Berdasarkan hasil dan pengujian, dapat ditarik kesimpulan bahwa infrastruktur analisis data *heart rate* dapat diimplementasikan menggunakan arsitektur *microservices*, berbasis docker *container* dan kubernetes untuk melakukan manajemen *cluster*. Pengujian *self-healing* menunjukkan bahwa sistem mampu kembali ke keadaan normal ketika terdapat *pod* yang mati. Pada pengujian *self-healing* juga didapatkan hasil bahwa ketika *pod* mati, *deployment* membuat *pod* baru terlebih dahulu dan menunggu hingga *pod* baru dalam keadaan berjalan, kemudian *pod* lama benar-benar dihapus dari *cluster*. Hal tersebut menunjukkan bahwa sistem akan selalu tersedia untuk diakses. Pengujian *persistent pod database* menunjukkan bahwa seluruh *pods database* dari setiap *microservices* mampu menyimpan data secara *persistent*. Pengujian skalabilitas menunjukkan terjadi peningkatan waktu yang drastis ketika melakukan analisis 50.000 data. Hasil pengujian analisis menggunakan skenario 100.000 data tidak mampu mengembalikan respon ke postman.

Berdasarkan penelitian yang telah dilakukan, terdapat beberapa saran yang dapat diberikan peneliti. Pertama perlu dilakukan optimasi terhadap *microservices heart rate analytic* agar mampu menangani proses analisis dengan data yang lebih besar dan dalam waktu yang lebih singkat. Kedua perlu dilakukan penambahan aplikasi *analytic* sehingga infrastruktur mampu meningkatkan kemampuan dalam melakukan analisis berbagai macam data. Ketiga perlu dilakukan penambahan fitur *service discovery* yang memungkinkan *microservices* dapat menemukan *microservices* lain secara otomatis. Keempat perlu dilakukan penambahan fitur keamanan, seperti mekanisme *autentikasi* untuk memastikan keamanan dari setiap *microservices*.

7. DAFTAR PUSTAKA

Adyandana, A., Pramukantoro, E. S., & Bakhtiar, F. A. (2019). Pengembangan Lightweight Data Analytic Pada Perangkat Edge.

Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M., & Steinder, M. (2016). Performance Evaluation of Microservices Architectures Using Containers.

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software, 33*(3), 42-52.

Bashir, M. R., & Gill, A. Q. (2016). Towards an IoT Big Data Analytics Framework: Smart Buildings Systems. *IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*.

Docker. (t.thn.). *docs.docker.com*. Dipetik December 19, 2019, dari https://docs.docker.com/engine/docker-overview/

Evans, D. (2011, Aprill). *How the Next Evolution of the Internet.* Dipetik January

19, 2019, dari https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

Faathin, A. R., Pramukantoro, E. S., & Bakhtiar, F. A. (2019). PENGEMBANGAN PERSONAL DATA ANALITIK MENGGUNAKAN PHP-ML DAN APACHE SPARK PADA IOT CLOUD APPS.

Jimangel. (2019). *https://kubernetes.io*. Dipetik July 26, 2019, dari https://kubernetes.io/docs/concepts/services-networking/connect-applications-service/

Kang, H., Le, M., & Tao, S. (2016). Container and Microservice Driven Design for Cloud Infrastructure DevOps.

Marathe, N., Gandhi, A., & Shah, J. M. (2019). Docker Swarm and Kubernetes in Cloud Computing Environment.

Marjani, M., Nasaruddin, F., Gani, A., Karim, A., Hashem, I. A., Siddiqa, A., & Yaqoob, I. (2017). Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges.

Naik, N. (2017). Docker container-based big data processing system in multiple clouds for everyone.

Pawar, K., & Attar, V. (2016). A Survey on Data Analytic Platforms for Internet of Things. *International Conference on Computing, Analytics and Security Trends (CAST)*, 605-610.

Shah, J., & Dubaria, D. (2019). Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform.

Smarvin. (2019). *https://kubernetes.io*. Dipetik July 26, 2019, dari https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

tddorgtfo. (2019). *https://kubernetes.io*. Dipetik July 26, 2019, dari https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/

Thönes, J. (2015). Microservices. *IEEE Software*, *32*(1), 116 - 116.

Vageesha17. (2019a). *https://kubernetes.io*. Dipetik July 26, 2019, dari https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/

Vageesha17. (2019b). *https://kubernetes.io/*. Dipetik December 6, 2019, dari https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Xu, P., Shi, S., & Chu, X. (2017). Performance Evaluation of Deep Learning Tools in Docker Containers.