P-ISSN : 1858-3938
E-ISSN : 2721-284X

PERBANDINGAN KINERJA API MICROSERVICE PADA JAVA SPRINGBOOT 3 MENGGUNAKAN GRAALVM DAN JVM PADA SERVER MESIN KUBERNETES

Muhammad Bagir[1]
[1]Sekolah Tinggi Teknologi Informasi NIIT / Program Studi Sistem Informasi, Jakarta Selatan

E-mail: [1] alchirrid@yahoo.com

Abstrak

Abstrak—API Microservices merupakan metode pengembangan aplikasi berbasis servis yang popular saat ini, Banyak platform dan Bahasa pemrograman menawarkan pengembangan API menggunakan Microservis, salah satunya Java. Java merupakan Bahasa pemrograman yang bebas platform, bisa berbasis sistem operasi Linux, Mac maupun Windows. Pengembangan dalam dunia Industri umumnya menggunakan Framework, seperti SpringBoot dan yang terbaru adalah SpringBoot 3. Dalam implementasinya SpringBoot 3 dengan Bahasa pemrograman Java menkonsumsi memori yang tidak kecil, sehingga cukup tinggi dalam hal biaya perawatan, khususnya yang diimplementasikan pada mesin cloud. Java Sendiri memiliki mesin JVM sebagai dasar dimana aplikasi dapat berjalan dan mengeksekusi perintah atau permintaan pengguna yang dapat berjalan pada Platfotm Sistem Operasi apapun. Penelitian ini bertujuan untuk melakukan perbandingan kinerja API Microservice dengan Java menggunakan Framework SpringBoot 3 pada Mesin Virtual pembanding yang berbeda, yakni menggunakan mesin native seperti GraalVM. Penelitian menggunakan Pendekatan komparatif kualitatif dengan melihat performa Aplikasi, besar memori yang dibutuhkan dan besar kemampuan servis menerima permintaan(request) proses dari pemanggil. Hasil Perbandingan nantinya akan menjadi rekomendasi layak tidaknya GraalVM menggantikan JVM pada mesin Kubernetes yang saat ini masih berjalan dan melayani jutaan transaksi. Dengan demikian penelitian ini dapat dijadikan pertimbangan bagi dunia industri yang menggunakan Java dan SpringBoot 3 dalam Sistem servisnya ke depan untuk beralih ke Native Mesin Virtual untuk kinerja Servis yang lebih optimal dan dapat mengurangi biaya.
Kata kunci: API, JVM, Java, SpringBoot 3, Native, GraalVM

Abstract

Abstract — Microservices API is a popular service-based application development method today, many platforms and programming languages offer API development using Microservices, one of which is Java. Java is a platform-free programming language, which can be based on Linux, Mac or Windows operating systems. Development in the Industrial world generally uses Frameworks, such as SpringBoot and the latest is SpringBoot 3. In its implementation, SpringBoot 3 with the Java programming language consumes a lot of memory, so it is quite high in terms of maintenance costs, especially those implemented on cloud machines. Java itself has a JVM engine as the basis where applications can run and execute commands or user requests that can run on any Operating System Platfotm. This research aims to compare the performance of the Microservice API with Java using the SpringBoot 3 Framework on different comparison Virtual Machines, namely using native machines such as GraalVM. The research uses a qualitative comparative approach by looking at application performance, the amount of memory required and the ability of the service to receive requests from the caller. The comparison result will be a recommendation whether or not GraalVM should replace JVM on the Kubernetes machine which is currently still running and serving millions of transactions. Thus this research can be used as a consideration for the industrial world..
Key word: *API, JVM, Java, SpringBoot 3, Native, GraalVM*

1. Pendahuluan

Java sebagai sebuah Bahasa pemrograman, banyak digunakan oleh praktisi dunia industri dalam bisnis skala besar hampir di seluruh dunia. Java hadir dalam banyak platform baik desktop, web, enterprise maupun mobile. Java juga hadir dalam banyak platform system operasi seperti linux, mac dan windows. Dan Java memiliki beberapa edisi antara lain :

- Java Card – Edisi *Smart Card*
- Java ME – Edisi *Micro*, banyak digunakan pada perangkat mobile.

P-ISSN : 1858-3938
E-ISSN : 2721-284X

- Java SE – *Standard Edition.*
- Java EE – *Enterprises Edition*

Untuk microservices, khususnya yang digunakan pada pengembangan aplikasi API (*Application Programming Interface*) edisi java yang digunakan adalah kombinasi *Standard Edition* dan *Enterprise Edition*. API dengan microservices membutuhkan lingkungan pengembangan dan implementasi yang besar karena digunakan untuk menangani transaksi berskala besar, dalam hal ini penerapan teknologi mesin Kubernetes Sangat dibutuhkan seperti Google Kubernetes. Pada implementasinya ada beberapa tantangan yang sering dihadapi para pelaku industri yang core bisnisnya sangat bergantung pada sukses tidaknya penggunaan teknologi informasi khususnya arsitektur API dan performasi dan servis dalam menangani banyaknya transaksi dengan minimum downtime.

SpringBoot merupakan salah satu teknologi framework pengembangan berbasis java yang sering digunakan dan paling secure saat ini dalam pengembangan API, versi 3 merupakan versi terakhir yang dirilis Spring pada tahun 2023. Java sendiri memiliki mesin virtual yang dikenal dengan JVM(Java Virtual Machine) yang membuatnya mampu berjalan di banyak platform sistem operasi, mesin ini yang bertugas menerjemahkan bytecode java menjadi Bahasa mesin. Saat menjalankan aplikasi JVM ini akan bekerja dan menggunakan Java Runtime Environment untuk bekerja dan mengeksekusi setiap permintaan dari pengguna aplikasi dan servis. Berbeda dengan JVM, juga terdapat mesin virtual yang dikembangkan oracle yang katanya dapat meningkatkan kinerja aplikasi berbasis Java dan JVM menggunakan Kompiler Just-in-time (JIT), menurunkan latensi aplikasi serta meningkatkan kinerja puncak dan mengurangi waktu yang habis digunakna untuk pengumpulan garbage. Selain itu juga dilengkapi dengan utility native image yang dapat mengkompile java bytecode serta menghasilkan aplikasi yang bersifat executable sehingga aplikasi dapat lebih cepat startupnya dan menggunakan alokasi memori yang lebih kecil.

Berangkat dari latar belakang ini penulis tertarik untuk membuktikan kinerja native virtual machine dalam hal ini graalVM dan membandingkannya dengan JVM dengan rumusan masalah Bagaimana Perbandingan Kinerja API Microservice pada Java SpringBoot 3 menggunakan GraalVM dan JVM pada mesin kubernetes (Google Kubernetes Engine)?

Penelitian ini bertujuan untuk memberikan manfaat kepada penulis, dosen, mahasiswa maupun pengembang dalam dunia industri dalam menentukan platform mesin virtual yang dapat digunakan untuk meningkatkan kinerja servis dan mengurangi biaya dengan menurunkan penggunaan memori.

Metode yang digunakan pada penelitian ini adalah komparatif dengan melihat perbandingan nilai berupa satuan waktu maupun besar transaksi dari instant atau pod pada servis yang dijalankan baik pada mesin virtual JVM maupun Native yang menggunakan GraalVM.

2. Metode Penelitian

Secara umum metode penelitian diartikan sebagai cara ilmiah untuk mendapatkan data dengan tujuan dan kegunaan tertentu. Jenis penelitian yang digunakan dalam penelitian ini adalah penelitian secara komparatif dengan pendekatan kuantitatif, dimana

Penelitian komparatif adalah penelitian yang membandingkan keadaan satu variabel atau lebih pada dua atau lebih sampel yang berbeda, atau dua waktu yang berbeda. [1]. Adapun penerapan penelitian komparatif pada penelitian ini digunakan untuk mengetahui perbandingan kinerja servis antara penggunaan Java Virtual Machine dengan Graal Virtual Machine untuk servis yang dikembangkan menggunakan SpringBoot 3 pada lingkungan Server Google Kubernetes.

A. *Java Virtual Machine*

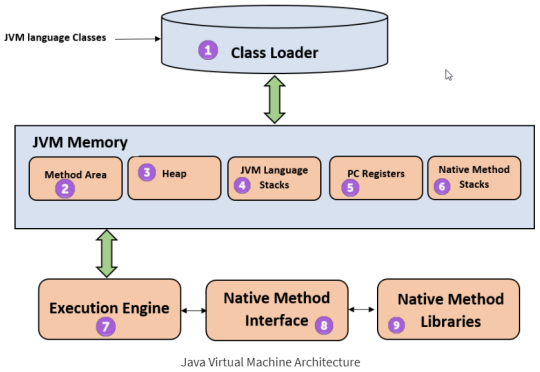
Java Virtual Machine adalah mesin virtual yang menyediakan lingkungan untuk berjalannya aplikasi java. *Java Virtual Machine* adalah landasan platform Java, adalah mesin komputasi abstrak. Seperti mesin komputasi nyata, ia memiliki kumpulan instruksi dan mengolah berbagai area pada memori saat runtime.[2]

Java Virtual Machine tidak tahu menahu tentang bahasa pemrograman Java, hanya mengetahui format biner saja dan format file kelas. File kelas berisi instruksi Mesin Virtual Java (atau bytecode).

Pada Spesifikasi *Java Virtual Machine*, perilaku dari virtual machine dapat dijelaskan dengan terminologi subsistem, area memori, tipe data dan instruksi. Komponen ini berguna untuk mendefinisikan perilaku eksternal dari implementasinya. Spesifikasinya mendefinisikan perilaku yang dibutuhkan dari setiap implementasi Java Virtual Machine untuk mengeksekusi komponen abstrak dan interaksinya.

Gambar 1 merupakan diagram blok Java Virtual Machine yang memasukkan subsistem utama dan area memori yang dijelaskan di spesifikasinya. Masing-masing Java Virtual Machine memiliki subsistem class loader, yakni sebuah mekanisme untuk memuat kelas maupun interface dengan ketentuan penamaan lengkapnya.

P-ISSN : 1858-3938
E-ISSN : 2721-284X



[1] Gambar 1. Arsitektur Java Virtual Machine (Sumber: https://www.guru99.com/difference-between-jdk-jre-jvm.html)

B. Graal Virtual Machine

GraalVM adalah virtual machine besutan oracle yang bersifat open-source dan memiliki performa sangat baik. Digunakan untuk mengkompilasi native image untuk meningkatkan kinerja startup aplikasi, juga mengurangi konsumsi memori dan ukuran file aplikasi berbasis Java.

GraalVM Native Image adalah suatu mekanisme eksekusi mandiri yang dihasilkan dari proses kompilasi yang dilakukan java. Bagi pengembang Java Spring Boot dapat dilakukan dengan menambahkan Profil Maven dengan plugin native-image

```
<plugin>
  <groupId>org.graalvm.buildtools</groupId>

<artifactId>native-maven-plugin</artifactId>
</plugin>
```

Pada Penelitian akan ini digunakan GraalVM Community dengan versi native-maven-plugin version 0.9.28 dan JDK versi 17.

C. API Microservice

API (*Application Programming Interface*) adalah sekumpulan perintah, fungsi, dan protokol yang dapat digunakan programmer saat membangun perangkat lunak untuk sistem operasi tertentu. API menyediakan fungsi dan perintah dengan bahasa yang terstruktur dan mudah difahami yang digunakan untuk pengembangan, API memudahkan pengembangan sistem.

Dengan adanya API maka perangkat lunak yang berinteraksi dengan perangkat lunak lainnya dapat berinteraksi melalui suatu aturan standar untuk mengakses data ataupun sumber data yang disediakan tanpa perlu tahu bagaimana aplikasi tersebut dibuat.

API terdiri dari beberapa komponen antara lain :

- *Function* (Fungsi)
- Protocol (Protokol)
- Data Structure (Struktur Data)
- Object (Objek)
- Parameter (Parameter)

API dapat diklasifikasikan menjadi sebagai berikut :

Tabel 1. Tabel Klasifikasi

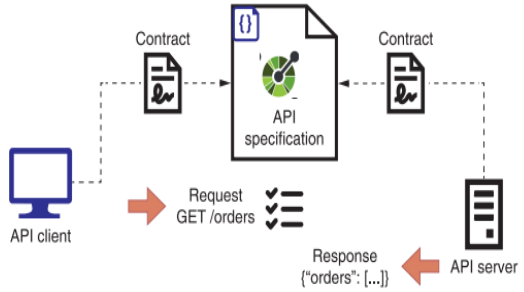
Kategori API	Deskripsi	Contoh
Sistem Operasi	API yang menangani fungsi dasar yang dapat dilakukan computer seperti I/O dan Eksekusi Program	API Sistem Operasi Microsoft Windows
Bahasa Pemrograman	API yang dapat digunakan untuk memperluas kapabilitas eksekusi suatu Bahasa pemrograman	Java API
Infrastructure Service (Layanan Infratruktur)	API ini digunakan untuk akses infrastruktur seperti media penyimpanan, aplikasi dan lain-lain.	*Amazon Elastic Compute Cloud* (Amazon EC2), *Google Kubernetes Engine*
Web & Application Service	API yang digunakan untuk mengakses data dan layanan yang disediakan suatu aplikasi dan web	API untuk SAP, API Facebook, API Google

Sementara Microservices adalah gaya arsitektur dimana komponen sistem dirancang sebagai aplikasi yang berdiri sendiri dan dapat

digunakan secara independen. Konsep microservice sudah ada sejak awal tahun 2000-an dan sejak 2010-an menjadi makin popular[4].

Microservices juga dapat didefinisikan secara berbeda sesuai dengan aspek yang mau kita tekankan pada Microservices tersebut, menggunakan kata micro yang menunjukkan bahwa ia adalah sesuatu yang kecil. Kecil disini bukan menunjukkan ukuran aplikasi atau lingkungan pengembangannya melainkan suatu gagasan bahwa Microservices adalah aplikasi dengan ruang lingkup yang didefinisikan dan sempit.

Microservices berkomunikasi dengan microservices lainnya melalui API, dalam hal ini API menjadi wakil interface Microservices. API harus memiliki dokumentasi, yang mana dokumentasi ini akan menginformasikan kepada kita apa yang harus dilakukan untuk berinteraksi dengan Microservices dan respon seperti apa yang didapatkan. Semakin baik dan lengkap dokumentasi API semakin jelas bagi pengguna mengenai cara kerja API tersebut. Dalam hal ini dapat dilihat gambar 2 yang menjelaskan kontrak antar servis.



Gambar 2. Spesifikasi API yang menggambarkan kontrak API Server dan Client

Berikut adalah kelebihan Microservices:[5]

- Untuk Aplikasi besar, membuat aplikasi terpisah menjadi komponen-komponen kecil yang memiliki tugas yang berbeda, dan dikerjakan secara mandiri.
- Setiap komponen dapat diskalakan secara independent.
- Implementasi dapat diubah tanpa mengganggu fungsi keseluruhan aplikasi selama interface tetap sama.
- Kode yang relatif lebih kecil dalam setiap komponen memudahkan untuk dilakukan peningkatan ataupun perbaikan yang lebih cepat sehingga aplikasi pun dapat tersedia dengan cepat dalam merespon kebutuhan bisnis.

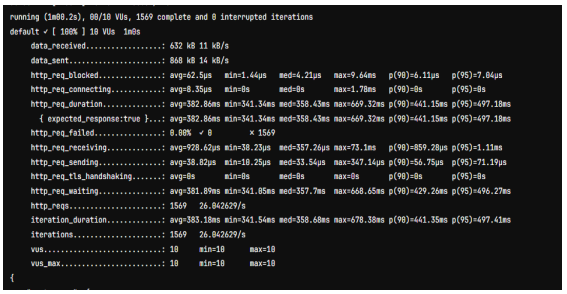
D. Mesin Kubernetes

Kubernetes adalah platform ¬open-source, portable yang dapat dikembangkan untuk pengelolaan container, dikembangkan oleh Google, Kubernetes atau yang disingkat K8s menjadikan seluruh rangkaian komputer dan membuatnya bekerja sebagai satu kesatuan. Dengan Kubernetes dapat dipastikan banyak orang yang dapat menggunakan aplikasi yang terkontainer tersebut dan dapat bekerja sama, walaupun mereka menggunakan platform yang berbeda. Dalam penelitian ini mesin Kubernetes yang digunakan adalah Google Kubernetes Engine (GKE)

Google Kubernetes Engine merupakan Manajemen layanan Kontainer yang disediakan Google melalui Google Cloud Platform(GCP) yang berbasis cloud. GKE adalah manajemen cluster dan sistem orkestrasi yang handal untuk menjalankan kontainer Docker yang dibangun menggunakan sistem Kubernetes Open Source.[6].

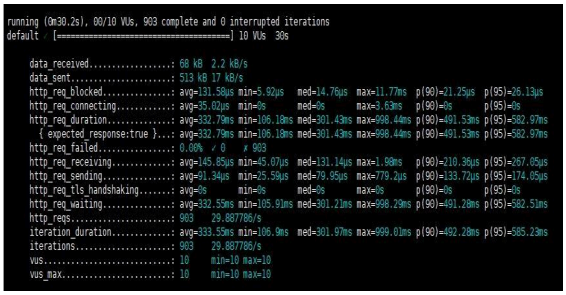
3. Hasil Penelitian dan Pembahasan

Hasil Penelitian diperoleh dengan melakukan *deployment* aplikasi servis menggunakan dua jenis mesin Virtual yakni JVM dan GraalVM pada lingkungan Google Kubernetes Engine. Pada penelitian ini akan digunakan 3 contoh Aplikasi servis, A, B dan C untuk load dengan kemampuan transaksi perdetik yang berbeda berdasarkan hasil load test di lingkungan GKE untuk masing-masing API. Masing-masing menggunakan 10VU(*Virtual User*). Untuk melihat hasil load test masing-masing transaksi tersebut dapat dilihat pada gambar 3 untuk servis A, Gambar 4 untuk Servis B dan Gambar 5 untuk servis C.

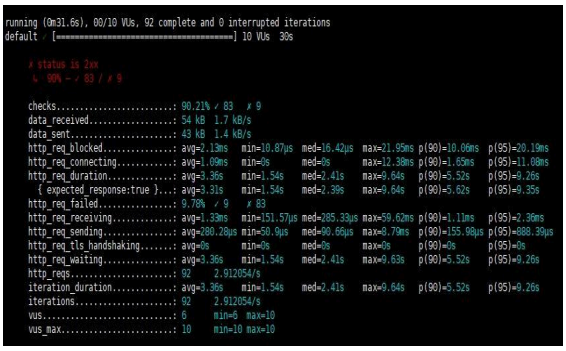


Gambar 3. Hasil Load Test Servis A

P-ISSN : 1858-3938
E-ISSN : 2721-284X



Gambar 4. Hasil Load Test Servis B



Gambar 5. Hasil Load Test Servis C

Dari hasil load test servis yang berjalan tersebut maka dapat diperoleh hasil seperti tabel dibawah ini:

Tabel 2. Kapasitas Transaksi/Servis/10VU/30 Detik

Nama Servis	Total Transaksi Selesai	Request Sukses (Transaksi)	Request Gagal (Transaksi)	Kemampuan Penanganan Transaksi/detik
Servis A	1569	1569	0	26
Servis B	903	903	0	29
Servis C	92	83	9	2

Masing-masing Servis tersebut pada implementasinya memiliki jumlah POD berbeda tergantung kebutuhan perharinya sesuai banyak transaksi dan kemampuan pelayanannya. Servis yang kemampuan penanganan transaksi perdetik sangat kecil membutuhkan instance pod yang lebih banyak jika ekspektasi bisnis untuk transaksi tersebut besar dan tentunya biaya juga tinggi.

Pada implementasinya juga terkadang ada servis yang bisa mengalami stuck, walaupun jarang sekali terjadi, hanya jika load transaksi besar dan servis mengalami glitch. Untuk servis yang mengalami stuck ini dibutuhkan restart servis, jika restart lama, maka tentunya down time juga menjadi lama, berikut adalah durasi lama servis untuk melakukan startup dapat dilihat pada gambar 6 untuk Servis A, Gambar 7 untuk Servis B dan Gambar 8 untuk Servis C.



Gambar 6. Startup Load Servis A



Gambar 7. Startup Load Servis B



Gambar 8. Startup Load Servis C

Dari Startup Deployment menggunakan JVM yang ditunjukkan gambar 6,7 dan 8 maka dapat dikelompokkan data waktu yang dibutuhkan untuk masing-masing servis hingga siap dialiri transaksi adalah:

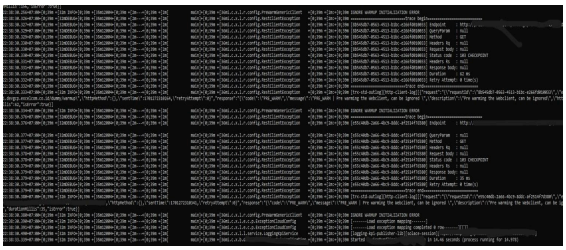
- Servis A : 77,749 detik
- Servis B : 171,711 detik
- Servis C : 133,487 detik

Dari hasil penelitian untuk masing-masing servis diatas dibutuhkan waktu yang tidak singkat untuk servis berada pada keadaan siap menerima request, hal ini akan berdampak cukup buruk bila servis harus direstart pada saat jam sibuk atau transaksi melampaui limit yang bisa dihandle 1 POD, contoh pada Servis B yang membutuhkan waktu startup hampir 3 menit. Sementara berdasarkan hasil load test servis B mampu menangani 29 transaksi perdetik, bayangkan jika terjadi transaksi ideal 29 transaksi perdetik, maka akan da 29 x 60 x 171,711 = 298.777,140/60 = 4979 transaksi yang tidak dapat diproses atau gagal. Berbeda kasusnya dengan transaksi yang bisa dihandle servis C yang hanya 2 transaksi perpod, jika dibandingkan servis B maka servis C 2 x 60 133,487 = 16.018,44/60=266 transaksi, Sementara itu untuk Servis A 26 x 60 x 77,749 =

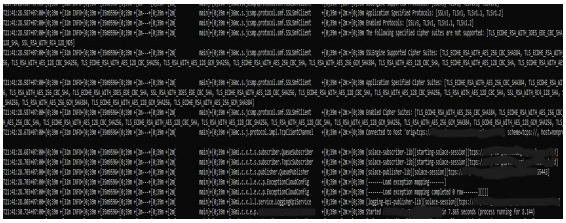
P-ISSN : 1858-3938
E-ISSN : 2721-284X

121.288,44/60 = 2021 transaksi untuk kasus ideal, jika transaksi yang terjadi pada saat servis tersebut dalam kondisi Stuck di Jam Sibuk dan membutuhkan restart.

Hasil ini harus dapat diminimalkan sehingga kemungkinan transaksi yang hilang atau gagal dapat diminimalisir, maka untuk meminimalisir kemungkinan terburuk yang terjadi pada servis di saat jam sibuk dan servis mengalami stuck, harus terjadi peningkatan kecepatan startup dari servis yang cukup signifikan.
Hasil tersebut akan kita bandingkan dengan startup menggunakan GraalVM untuk masing-masing Servis A, Servis B dan Servis C yang dapat dilihat pada gambar 9, 10 dan 11.



Gambar 9. Startup Load Servis A (GraalVM)



Gambar 10. Startup Load Servis B (GraalVM)



Gambar 11. Startup Load Servis C (GraalVM)

Dari hasil deployment menggunakan GraalVM, didapatkan hasil yang dapat dikelompokkan berdasarkan data waktu yang dibutuhkan servis untuk startup hingga siap dialiri transaksi adalah :

- Servis A : 14,978 detik
- Servis B : 7,865 detik
- Servis C : 15,837 detik

Dari Hasil uji coba terjadi perbedaan waktu startup yang signifikan untuk masing-masing microservis yang dideploy dan digunakan, dengan selisih waktu Startup untuk deploy menggunakan JVM menjadi sebesar :

A. 77,749-14,978 = 62,771 detik
B. 171,711-7,865= 163,846 detik
C. 133,487-15,837=117,65 detik

Berdasarkan efisiensi waktu yang diperoleh, maka diperoleh efisiensi waktu startup lebih dari 1 menit, bahkan untuk servis B mendekati 3 menit dan jika dihitung performa transaksi masing-masing servis terjadi peningkatan efiensi transaksi sebesar :

A. 26 x 62,771= 1632 transaksi
B. 29 x 163,846 = 4751 transaksi
C. 2 x 117,65 = 235 transaksi

4. Kesimpulan

Dari hasil penelitian dan pembahasan di atas, maka dapat disimpulkan hal-hal sebagai berikut:

1. Terdapat Perbedaan waktu startup yang signifikan penggunaan GraalVM dengan SpringBoot pada Google Kubernetes Engine.
2. Kinerja API Microservice dengan Java SpringBoot3 menggunakan GraalVM pada mesin Kubernetes lebih baik jika dibandingkan menggunakan JVM..

5. Referensi

Sugiyono, “Metode Penelitian Bisnis (Pendekatan Kuantitatif, Kualitatif)”, Edisi 3, Alfabeta, Bandung. 2018

L. Tim, Y. Frank, B. Gilad, Buckley. A dan S. Daniel, “The Java® Virtual Machine Specification Java SE 17 Edition”, Final Release, September 2021.

https://www.guru99.com/difference-between-jdk-jre-jvm.html

P. Jose Haro, “Microservice API Using Phython, Flask, FastAPI, OpenAPI and More”, Manning Publication, Shelter Island, 2023.

S.Anton et al, “Pedoman Keamanan Microservice Dan Application Programming Interface (API)”, Badan Siber dan Sandi Negara, Jakarta Selatan, Agustus 2021.

P. Odi dan C.D.Widiyanto, “Provisioning Google Kubernetes Engine Cluster Dengan Menggunakan Terraform Dan Jenkins Pada Dua Environment”, JIPI (Jurnal Ilmu Penelitian dan Pembelajaran Informatika), Vol. 8, Hal 597-606, Juni 2023