ISSN (p) : 2407-1323
ISSN (e) : 2442-4404
DOI : https://doi.org/ 10.25124/jett.v10i2.6603
| Vol. 10 | No. 2 | Halaman : 136 - 146

JETT Jurnal Elektro Telekomunikasi Terapan Desember 2023

PERFORMANSI DESAIN ORKESTRASI KUBERNETES MENGGUNAKAN CONTAINER RUNTIME

KUBERNETES ORCHESTRATION DESIGN PERFORMANCE USING CONTAINER RUNTIME

Bongga Arifwidodo[1], Jafaruddin Gusti Amri Ginting[2]

[1,2]Institut Teknologi Telkom Purwokerto
[1]bongga@ittelkom-pwt.ac.id,[2]jafaruddin@ittelkom-pwt.ac.id

Abstrak
Container merupakan salah satu evolusi teknologi virtualisasi yang banyak digunakan untuk mengembangkan dan mengirimkan perangkat lunak untuk layanan komputasi awan. Container memungkinkan pembagian sumber daya langsung pada host secara efisien antar platform penyewa cloud, sehingga lebih ringan dan lebih cepat daripada mesin virtual. Container runtime dalam arsitektur container, bertanggung jawab memuat image container dari repositori, memantau sumber daya sistem lokal, mengisolasi sumber daya sistem untuk penggunaan container, dan mengelola siklus hidup container. Host dengan jumlah container yang banyak menyebabkan penurunan performansi yang signifikan, baik container yang dipasangkan pada platform penyewa cloud maupun pada local host. Masalah ini muncul karena container runtime bertanggung jawab menjalankan container pada host sistem operasi. Pemilihan container runtime menjadi sangat krusial untuk mengimbangi pertambahan jumlah container yang digunakan. Pada penelitian ini berfokus menganalisis performansi dari tiga container runtime yaitu Containerd, CRI-O, dan Kata Containers pada orkestrasi Kubernetes. Skenario penelitian ini menggunakan skalabilitas dengan jumlah container yang berbeda yaitu 10, 20, dan 40 container. Parameter yang akan dianalisis yaitu performansi dari CPU, memory, throughput, dan latency. Berdasarkan hasil pengujian diketahui bahwa container runtime Containerd memiliki kinerja yang unggul dan optimal berdasarkan hasil dari parameter throughput, latency, dan memory dengan hasil masing-masing parameter yaitu 2301,75 MB/s, 0,359 ms, dan 13001,8 MB/s.

Kata kunci: Container runtime, ContainerD, CRI-O, Kata Containers, Kubernetes

Abstract
Containers are one of the evolutions of virtualization technology that is widely used to develop and deliver software for cloud computing services. Containers enable efficient sharing of resources directly on the host between cloud tenant platforms, making them lighter and faster than virtual machines. The container runtime in the container architecture, is responsible for loading container images from the repository, monitoring local system resources, isolating system resources for container usage, and managing the container lifecycle. Hosts with a large number of containers cause a significant decrease in performance, both containers deployed on cloud tenant platforms and on local hosts. This issue occurs because the container runtime is responsible for running the containers on the host operating system. The choice of container runtime is very crucial to keep up with the increasing number of containers used. This research focuses on analyzing the performance of three container runtimes namely Containerd, CRI-O, and Kata Containers in Kubernetes orchestration. This research scenario uses scalability with a different number of containers, namely 10, 20, and 40 containers. The parameters to be analyzed are the performance of the CPU, memory, throughput, and latency. Based on the test results it is known that the container runtime Containerd has superior and optimal performance based on the results of throughput, latency, and memory parameters with the results of each parameter being 2301.75 MB/s, 0.359 ms, and 13001.8 MB/s.

Keywords: Container runtime, Containerd, CRI-O, Kata Containers, Kubernetes

1. PENDAHULUAN

Teknologi IT semakin berkembang, sejalan dengan prinsip efisiensi keuangan dengan konsekuensi pada penghematan sumber daya. Teknologi untuk penghematan daya salah satunya

ISSN (p) : 2407-1323
ISSN (e) : 2442-4404
DOI : https://doi.org/ 10.25124/jett.v10i2.6603
| Vol. 10 | No. 2 | Halaman : 136 - 146

adalah virtualisasi. Tidak jauh berbeda dengan fungsi aslinya, fungsi teknologi virtualisasi untuk membuat sebuat system virtual yang berfungsi seperti sistem asli [1]. Teknik umum virtualisasi dalam membangun teknologi virtualisasi, yaitu virtualisasi berbasis *hypervisor* dan virtualiasasi berbasis *container* [2]. Container adalah perkembangan teknologi virtualisasi yang banyak digunakan untuk mengembangkan dan mengirimkan perangkat lunak untuk layanan cloud computing [3]. Teknologi container memiliki peran utama dalam perubahan ini yang memungkinkan untuk berbagi sumber daya host secara efisien [4]. Dalam arsitektur container, container runtime bertanggung jawab untuk memuat image container dari repositori, memantau sumber daya sistem lokal, mengisolasi sumber daya sistem untuk penggunaan container, dan mengelola siklus hidup container [5].

Penelitian [4] tentang performansi dari 2 (dua) runtime yang umum digunakan yaitu containerd dan CRI-O pada 2 (dua) Open Container Initiative (OCI) yang berbeda yaitu runc dan gVisor. Hasil performa yang diperoleh menunjukkan performa Containerd yang lebih baik dalam hal penggunaan CPU time sebesar 23,86 s, latensi memori sebesar 0,11 ms, dan aspek skalabilitas sebesar 37,69 s pada 50 kontainer. Sedangkan operasi sistem file (khususnya operasi write) dilakukan lebih efisien oleh CRI-O sebesar 2,43 s. Penelitian [5] meneliti mengenai 2 (dua) runtime yang berbeda yaitu Crun dan Kata Containers. Pengujian runtime menggunakan tools Python yang disebut pyperformance dengan 20 kali percobaan untuk setiap runtime. Dengan tipe parameter pada Python yang diuji yaitu Apps, Math, Logging, SciMark, Serialize, SQL, SymPy, Regex, Template, dan Various Hasil secara keseluruhan yaitu Crun memiliki rata-rata performansi 16 % lebih baik dari Kata Containers. Penelitian [6] meneliti mengenai perbandingan kinerja OCI menggunakan beberapa container engine yaitu Docker 18, Docker 20, Rkt, WSL2, Podman, Firecracker, Kata Containers, dan gVisor. Hasilnya Kinerja CPU pada Docker dan Podman lebih bagus.

Berbeda dengan sebelumnya, penelitian ini menggunakan container runtime yaitu ContainerD, CRI-O, dan Kata Containers untuk melihat perfomansinya dalam orkestrasi Kubernetes. Container runtime ini juga merupakan improvement dari penelitian sebelumnya karena penelitian ini menguji container runtime dari kategori high-level dan low-level dengan menggunakan jenis komunikasi intra-cluster. Komunikasi pod dalam penelitian ini menggunakan *intra-cluster* yakni komunikasi yang berada di dalam cluster Kubernetes antara pod (container) satu dengan pod (container) lainnya. Skenario pada penelitian [4] menggunakan jumlah container 5, 10, dan 50 container, sedangkan penelitian penulis menggunakan jumlah container 10, 20, dan 40 container dengan jumlah percobaan 30 kali agar memberikan gambaran performasi yang optimal. Parameter performansi yaitu parameter CPU dan memory, serta menambahkan parameter throughput, dan latency.

2. PERANCANGAN SISTEM

2.1 Container

Container menyediakan lingkungan yang ringan dan terisolasi yang membuat aplikasi lebih mudah dikembangkan, disebarkan, dan dikelola [7]. Adanya container membuat aplikasi *user* menjadi mudah dikomunikasikan dan dapat berjalan pada perangkat yang lain [8]. Timbul masalah saat melakukan manajemen aplikasi atau program pada jaringan yang sama, untuk mengurangi masalah tersebut pada container, aplikasi dipisahkan dari lingkungan asal aplikasi oleh container [9].

2.2 Container Runtime

ISSN (p) : 2407-1323
ISSN (e) : 2442-4404
DOI : https://doi.org/ 10.25124/jett.v10i2.6603
| Vol. 10 | No. 2 | Halaman : 136 - 146

Merupakan perangkat lunak yang mempunyai tanggung jawab menjalankan container dan mengelola image container pada node *deployment* [4]. Container runtime mempunyai 2 (dua) kategori yaitu container runtime low-level yang mematuhi spesifikasi runtime Open Container Initiative (OCI), dan container runtime high-level yang sesuai dengan Container Runtime Interface (CRI) spesifikasi runtime. OCI adalah proyek Linux Foundation, terutama berfokus pada penentuan panduan, standar, dan spesifikasi untuk Linux container. Spesifikasi OCI Runtime sebagian besar berurusan dengan pengelolaan siklus hidup container dan konfigurasi untuk berbagai platform, seperti Linux, Windows, dan Solaris. Container runtime yang sesuai dengan spesifikasi OCI dianggap sebagai runtime "low-level" [10]. Pada node Kubernetes, container runtime terdapat di lapisan terbawah yang berfungsi untuk memulai dan menghentikan container. Sebagai bagian dari upaya untuk membuat Kubernetes lebih dapat diperluas, Kubernetes telah mengerjakan API plugin baru untuk container runtime di Kubernetes, yang disebut Container Runtime Interface (CRI) [11]. Standar komunikasi antarmuka untuk plugin Kubernetes yang menjalankan dan mengawasi container adalah *Container Runtime Interface* (CRI).

A. ContainerD

ContainerD adalah runtime default yang digunakan oleh Docker Engine. Containerd menyimpan dan mengelola gambar dan snapshot; itu memulai dan menghentikan container dengan mendelegasikan tugas eksekusi ke runtime OCI. Untuk memulai proses containerisasi baru, containerd harus melakukan tindakan berikut: 1) membuat Container baru dari OCI image yang diberikan; 2) buat Task (tugas) baru dalam konteks Container; 3) mulai Tugas (pada titik ini runc mengambil alih dan mulai menjalankan bundel OCI yang disediakan oleh containerd).

B. CRI-O

CRI-O adalah container runtime yang dibangun untuk menjembatani antara runtime OCI dan CRI Kubernetes tingkat tinggi. Ini didasarkan pada versi arsitektur Docker yang lebih lama yang dibangun di sekitar driver grafik. Runtime CRI-O memanfaatkan Open Container Initiative (OCI), yang menyediakan spesifikasi untuk konfigurasi, lingkungan eksekusi, dan siklus hidup sebuah container serta spesifikasi untuk konfigurasi, sistem file, indeks, dan manifes *image*.

C. Kata Containers

Kata Containers adalah komunitas open source yang bekerja untuk membangun container runtime yang aman dengan mesin virtual yang lebih ringan dan berkerja seperti container [12]. untime Kata Containers kompatibel dengan spesifikasi runtime OCI dan oleh karena itu bekerja mulus dengan Container Runtime Interface (CRI) Kubernetes melalui implementasi CRI-O dan containerd [13].

2.3 Kubernetes

Kubernetes merupakan orkestrasi container yang juga dikenal dengan K8s. Tanggung jawab manajemen container Kubernetes meliputi penyebaran container, penskalaan (scaling dan descaling) container, serta load balancing container [14]. Kubernetes adalah framework open-source yang dibuat di lab Google untuk mengawasi aplikasi dalam container berbagai macam situasi, misalnya fisik, virtual, dan cloud framework [14].

2.4 Parameter QoS

ISSN (p) : 2407-1323
ISSN (e) : 2442-4404
DOI : https://doi.org/ 10.25124/jett.v10i2.6603
| Vol. 10 | No. 2 | Halaman : 136 - 146

Quality of Service sering disebut dengan QoS merupakan sebuah layanan yang melakukan pengukuran untuk mengetahui kualitas suatu jaringan [15].

A. Throughput

Throuhgput adalah besarnya nilai kecepatan transfer data yang diukur dalam satuan bps (bit per second). Klasifikasi standarisasi nilai Throughput berdasarkan TIPHON TR 101 329 V2.1.1 (1999-06) diuraikan pada Tabel 1.

$$Throughput : \frac{Dp}{Ld} \qquad (1)$$

dimana :
Dp = total data yang dikirim
Ld = total paket yang diterima

Tabel 1 Standar Throughput

Kategori	Throughput
Sangat Bagus	76 s/d 100 kbps
Bagus	51 s/d 75 kbps
Kurang Bagus	26 /d 50 kbps

B. Latency

Jenis latency yang diukur pada penelitian ini adalah latency pada jaringan. Latensi jaringan adalah jumlah waktu yang diperlukan paket data untuk berpindah dari satu tempat ke tempat lain [15]. Latency adalah jumlah waktu yang diperlukan paket data untuk berpindah dari satu titik di jaringan ke titik lainnya. Menurunkan latensi adalah bagian penting dalam membangun user experience yang baik [16].

$$Latency: \frac{Td}{Tp} \qquad (2)$$

dimana :
Td = total delay
Tp = total paket yang diterima

Tabel 2 Standar Latency

Kategori	Latency
Sangat Bagus	< 150 ms
Bagus	150 ms – 300 ms
Kurang Bagus	300 ms – 450 ms

2.5 Parameter Kinerja Hardware

A. CPU usage

CPU mewakili pemrosesan komputasi dan ditentukan dalam satuan CPU Kubernetes. Batasan dan permintaan sumber daya CPU diukur dalam unit cpu. Di Kubernetes, 1 unit CPU setara dengan 1 inti CPU fisik, atau 1 inti virtual, bergantung pada apakah node tersebut adalah host fisik atau mesin virtual yang berjalan di dalam mesin fisik. CPU diukur menggunakan tools y-cruncher. Y-cruncher merupakan program CPU benchmark yang dihitung dalam satuan Pi (banyaknya konstanta atau perintah yang dijalankan). Satuan Pi ini akan dikonversi ke satuan persentase (%) agar mempermudah proses analisis.

ISSN (p) : 2407-1323
ISSN (e) : 2442-4404
DOI : https://doi.org/ 10.25124/jett.v10i2.6603
| Vol. 10 | No. 2 | Halaman : 136 - 146

$$CPU\ (\%) : \frac{JPpi}{Tpi} \text{ x } 100\ \% \quad (3)$$

dimana :
JPpi = jumlah Pi yang dapat diproses
Tpi = total Pi

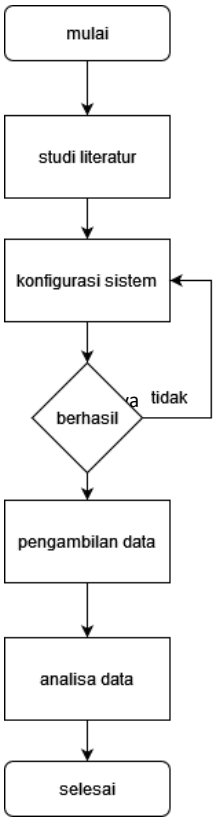
C. Memory

CPU komputer semakin cepat jauh lebih cepat daripada sistem memori komputer. Seiring perkembangan ini, semakin banyak program akan dibatasi kinerjanya oleh bandwidth memori sistem, bukan oleh kinerja komputasi CPU 1 [16].

$$memory : \frac{mU}{Tt} \quad (4)$$

dimana :
mU = banyaknya memori yang digunakan (MB)
Tt = total waktu (s)

2.6 Alur Penelitian

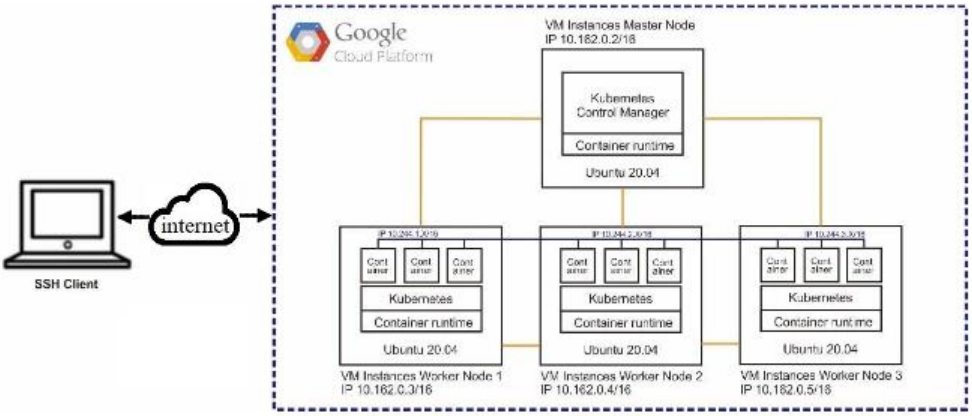


Gambar 1. Alur penelitian

2.7 Topologi Penelitian

ISSN (p) : 2407-1323
ISSN (e) : 2442-4404
DOI : https://doi.org/ 10.25124/jett.v10i2.6603
| Vol. 10 | No. 2 | Halaman : 136 - 146

Topologi pada Gambar 2 terbagi dalam area Google Cloud Platform dan area client. Google Cloud Platform berfungsi untuk membuat virtual mesin (VM) instances. Internal IP berfungsi sebagai IP cluster yang digunakan untuk komunikasi antar worker node dan master node.



Gambar 2. Topologi penelitian

2.8 Skenario Penelitian

Tabel 3 merupakan skenario yang digunakan dalam penelitian. Skenario terbagi menjadi 3 dengan ketiga container runtime, Containerd, CRI-O, dan Kata Containers. Skenario skalabilitas menggunakan jumlah container yang dipasangkan secara bergantian dengan jumlah 10, 20 dan 40 dengan 30 kali percobaan.

Tabel 3 Skenario

S	Container	Jumlah	Jumlah
1	ContainerD	10	30
		20	
		40	
2	CRI-O	10	30
		20	
		40	
3	Kata	10	30

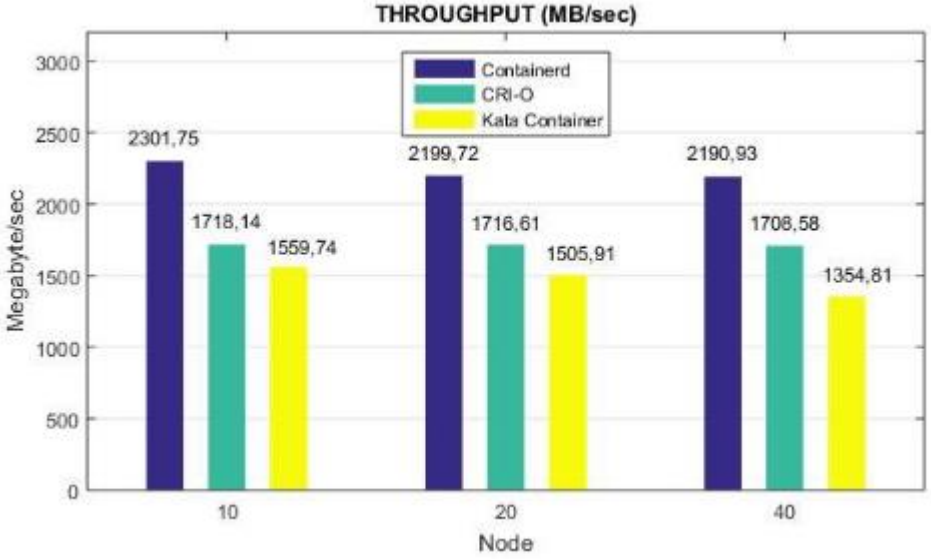
3.HASIL DAN PENGUJIAN SISTEM

3.1 Parameter Throughput

Hasil data throughput diambil menggunakan tools Netperf. Hasil menunjukkan setiap percobaan dengan jumlah container yang berbeda, akan mempengaruhi nilai throughput yakni mengalami penurunan nilai saat terjadi penambahan jumlah node container. Containerd memiliki nilai throughput tertinggi dibanding container runtime CRI-O dan Kata Containers. Hal itu karena pada Containerd menggabungkan antara interface runtime secara langsung ke dalam arsitektur Containerd, di mana hal ini menyebabkan proses komunikasi antara container menjadi lebih ringkas sehingga meningkatkan performansi throughput pada Containerd. Sedangkan untuk runtime CRI-O setelah pembuatan container runtime, CRI-O memerlukan setup conmon yang berfungsi sebagai program pemantauan dan alat komunikasi antar container. Di mana CRI-O memerlukan 1 (satu) step lebih banyak dari Containerd untuk melakukan komunikasi antar container.

ISSN (p) : 2407-1323
ISSN (e) : 2442-4404
DOI : https://doi.org/ 10.25124/jett.v10i2.6603
| Vol. 10 | No. 2 | Halaman : 136 - 146

Gambar 3 menunjukkan grafik hasil data throughput dengan nilai rerata 30 percobaan yang dilakukan. Terlihat nilai tertinggi didapatkan oleh container runtime Containerd pada pengujian 10 container dengan nilai throughput sebesar 2301,75 MB/sec. Nilai terendah oleh container runtime Kata Containers pada pengujian 40 container dengan nilai throughput sebesar 1354,81 MB/sec. Berdasarkan standar TIPHON, hasil data throughput yang didapatkan pada penelitian ini termasuk dalam kategori sangat baik karena nilainya > 2,1 MBps.

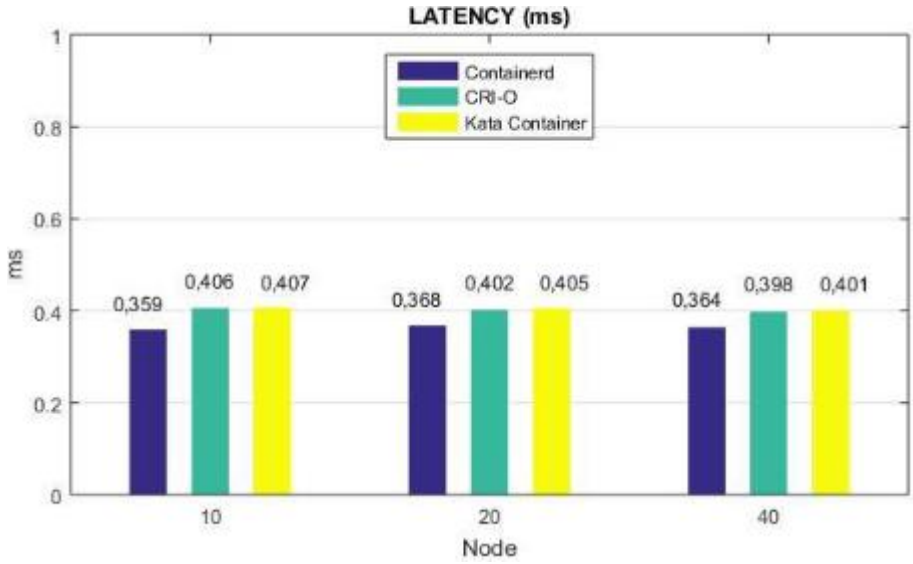


Gambar 3. Hasil Throughput

3.2 Parameter Latency

Container runtime Containerd memiliki nilai latency yang lebih kecil dibanding CRI-O dan Kata Containers karena pada Containerd menggabungkan interface runtime secara langsung ke arsitektur container. Hal inilah yang menyebabkan proses komunikasi antar container menjadi lebih singkat sehingga hasil latency juga rendah dan performa meningkat. Container Runtime CRI-O, memerlukan *setup common* yang berfungsi sebagai program pemantauan dan alat komunikasi antar container setelah pembuatan container runtime. Sedangkan hasil latency pada runtime Kata Container memiliki nilai yang rendah karena Kata Containers memiliki isolasi jaringan yang khusus. Kata Containers akan memastikan setiap trafik keluar dan masuk ke container adalah trafik yang berasal dari komunikasi antar container (bukan dari luar container) yang dapat menyebabkan nilai latency yang tinggi.

ISSN (p) : 2407-1323
ISSN (e) : 2442-4404
DOI : https://doi.org/ 10.25124/jett.v10i2.6603
| Vol. 10 | No. 2 | Halaman : 136 - 146



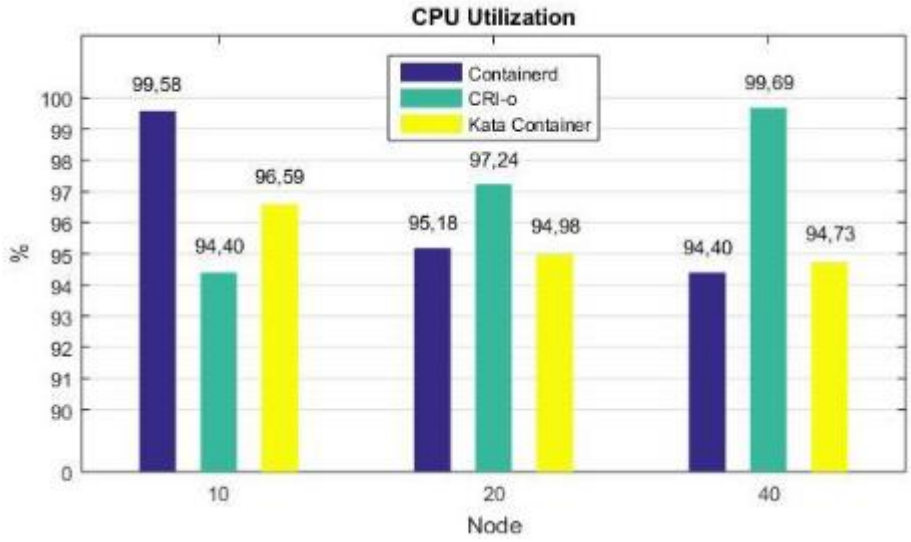
Gambar 4. Hasil Latency

3.3 Parameter CPU Usage

Penelitian ini menggunakan tools y-cruncher, diambil nilai dari CPU ulitization dan Multi-core Efficiency untuk hasil CPU. CPU utilization mengacu pada penggunaan sumber daya pemrosesan komputer, atau jumlah pekerjaan yang ditangani oleh CPU. Berbeda dengan pengukuran CPU utilization pada umumnya, y-cruncher menilai CPU utilization dengan seberapa maksimum load persentase kinerja yang dihasilkan, bukan persentase terendah. Cara kerja y-cruncher mengambil nilai tertinggi sebagai kinerja CPU yang bagus, karena y-cruncher merupakan salah satu benchmark tools yang membandingkan kinerja CPU dengan berbagai macam jenis spesifikasi menggunakan nilai persentase tertinggi dan juga nilai yang berhasil dicapai oleh CPU dalam memproses data.

Terlihat nilai tertinggi didapatkan oleh container runtime CRI-O pada pengujian 40 container dengan nilai CPU utilization sebesar 99,69%. Sedangkan nilai terendah didapatkan oleh container runtime Containerd pada pengujian 40 container dan CRI-O pada 10 container dengan nilai CPU utilization sebesar 94,90 %. Containerd unggul pada skenario jumlah container yang sedikit, karena Containerd memiliki konfigurasi CRI yang lebih ringkas sehingga akan lebih optimal ketika menggunakan jumlah kontainer yang sedikit, namun akan menurun kinerjanya ketika menggunakan jumlah container yang semakin banyak. Sedangkan CRI-O unggul di skenario jumlah container yang banyak karena CRI-O memiliki konfigurasi CRI yang lebih lengkap untuk pertambahan jumlah container. Tetapi akan mengalami performansi yang menurun ketika menggunakan sedikit container.

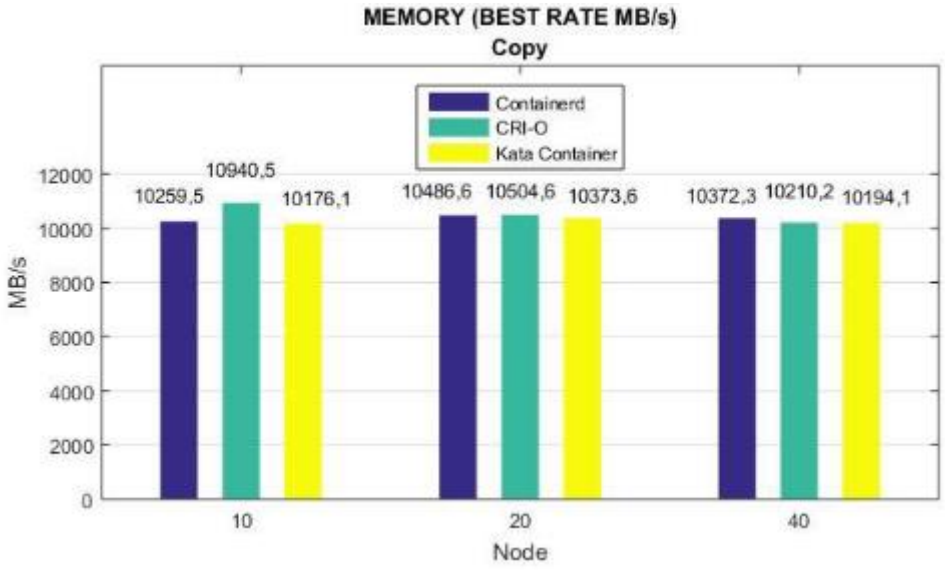
ISSN (p) : 2407-1323
ISSN (e) : 2442-4404
DOI : https://doi.org/ 10.25124/jett.v10i2.6603
| Vol. 10 | No. 2 | Halaman : 136 - 146



Gambar 5. Hasil CPU usage

3.4 Hasil Parameter Memory

Terlihat nilai tertinggi didapatkan oleh container runtime CRI-O pada pengujian 10 container dengan nilai memory sebesar 10940,5 MB/sec. Sedangkan nilai terendah didapatkan oleh container runtime Kata Containers pada pengujian 10 container dengan nilai memory sebesar 10176,1 MB/s. Pengukuran pada kernel vector operasi copy menggunakan STREAM menunjukkan bahwa CRI-O unggul disebagian besar skenario penambahan jumlah container, kecuali pada jumlah 40 container. Pengukuran kernel copy merupakan perpindahan data ke suatu sistem. Pada hasil memory (copy), runtime CRI-O unggul karena CRI-O berjalan pada abstraksi CRI yang merupakan abstraksi container runtime high level. Di mana CRI-O menjalankan container secara langsung menggunakan abstraksi CRI. Sedangkan pada container runtime Kata Container menggunakan abstraksi OCI yang merupakan runtime low-level.



Gambar 6. Hasil Memory

ISSN (p) : 2407-1323
ISSN (e) : 2442-4404
DOI : https://doi.org/ 10.25124/jett.v10i2.6603
| Vol. 10 | No. 2 | Halaman : 136 - 146

4.KESIMPULAN

Berdasarkan data yang sudah diperoleh dan dianalisa, Container runtime yang memiliki kinerja optimal pada platform Kubernetes adalah container runtime kategori high-level yaitu runtime Containerd dengan hasil parameter throughput 2301,75 MB/s, latency 0,359 ms, dan memory (scale) 13001,8 MB/s, serta runtime CRI-O dengan hasil parameter CPU utilization 99,69 %, dan memory (copy) 10940,5 MB/s. Pengaruh skalabilitas kontainer pada container runtime mendapatkan hasil bahwa Kata Containers memiliki hasil yang kurang optimal dibanding kontainer runtime Containerd dan CRI-O untuk seluruh parameter.

DAFTAR PUSTAKA

[1] A. &. S. Y. H. Widarma, "Analisis Kinerja Teknologi Virtualisasi Server (Study," dalam *Seminar Nasional Multi Disiplin Ilmu (SEMNASMUDI)*, Kisaran, 2019.

[2] I. N. &. K. I. F. Kurniawan, " Implementasi Virtualisasi Menggunakan Xen Hypervisor," *Jurnal Manajemen Informatika*, vol. 6, no. 1, p. 36–42, 2016.

[3] I. M. a. H. Karatza, "Orchestrated Sandboxed Containers Unikernels and Virtual Machines for Isolation-enhanced Multitenant Workloads and Serverless Computing in Cloud," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 9, pp. 1-16, 2021.

[4] A. J. V. P. a. M. G. L. Espe, "Performance Evaluation of Container Runtimes," dalam *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, 2020.

[5] F. Björklund, A Comparison Between Native and Secure Runtimes, Skövde: University of Skövde, 2021.

[6] R. D. a. W. K. Hidouci, "Containers Runtimes War: A Comparative Study," dalam *Future Technologies Conference 2020*, Vancouver (Canada), 2020.

[7] Microsoft, ""Windows dan Kontainer," Microsoft," Microsoft, 2023. [Online]. Available: https://learn.microsoft.com/id- id/virtualization/windowscontainers/about/. [Diakses 30 Maret 2023].

[8] AWS, "Containers vs. Virtual Machines - Apa Perbedaan Antara Kontainer dan Mesin Virtual?," AWS Amazon, [Online]. Available: https://aws.amazon.com/id/compare/the-difference-between-containers- and-virtual-machines. [Diakses 30 Maret 2023].

[9] A. Ganne, "Cloud Data Security Methods: Kubernetes VS Docker Swarm," *International Research Journal of Modernization in Engineering Technology and Science*, vol. 4, no. 12, pp. 807-811, 2022.

[10] J. Clark, " "Container Runtimes | Kubernetes Guide and Tutorial," ContainIQ," 24 8 2022. [Online]. Available: https://www.containiq.com/post/container- runtimes. [Diakses 2 3 2023].

[11] Y.-J. Hong, "Introducing Container Runtime Interface (CRI) in Kubernetes," Kubernetes, 16 12 2016. [Online]. Available: https://kubernetes.io/blog/2016/12/container-runtime-interface-cri-in- kubernetes/. [Diakses 3 4 2023].

[12] K. Containers, "About Kata Containers," Kata Containers, 2023. [Online]. Available: https://katacontainers.io/. [Diakses 2 4 2023].

[13] GabyCT, "Kata Containers Architecture," GitHub, 30 Maret 2023. [Online]. Available: https://github.com/kata-containers/kata-containers/tree/main/docs/design/architecture. [Diakses 4 April 2023].

[14] A. G. a. J. M. S. N. Marathe, "Docker Swarm and Kubernetes in Cloud Computing," dalam *3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, Tirunelveli, India, 2019 .

ISSN (p) : 2407-1323
ISSN (e) : 2442-4404
DOI : https://doi.org/ 10.25124/jett.v10i2.6603
| Vol. 10 | No. 2 | Halaman : 136 - 146

[15] ETSI, "Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); General aspects of Quality of Service (QoS)," dalam *Etsi Tr 101 329 V2.1.1, vol. 1*, 2020, p. 1–37.

[16] D. o. C. S. U. o. Virginia, "What is STREAM?," 2020. [Online]. Available: https://www.cs.virginia.edu/stream/ref.html. [Diakses 4 April 2023].

[17] J. S. a. D. Dubaria, "Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform," dalam *IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2019 .