BAB II
TINJAUAN PUSTAKA DAN LANDASAN TEORI

2.1 Tinjauan Pustaka

Penelitian terkait *Docker* dan *Podman* telah banyak dilakukan, sehingga dilakukan kajian Pustaka dengan tujuan memudahkan dalam menganalisis topik yang pernah dibuat untuk kemudian dijadikan sebagai referensi dalam penelitian ini.

Penelitian [13] yang dilakukan oleh Chairul Mukmin, Therezia Naraloka, Qodri Harun Andriyanto pada tahun 2021 yang berjudul "Analisis Perbandingan Kinerja Layanan *Container As A Service* (CAAS) Studi Kasus : *Docker* dan *Podman*" yaitu melakukan pengujian kinerja CPU, RAM, dan waktu respons pada web *server* menggunakan layanan CAAS dengan kondisi normal, kondisi menjalankan banyak aplikasi, kasus serangan DOS dan kasus penerapan pencegahan serangan DOS menggunakan teknik PPDIOO. Hasil penelitian menunjukkan bahwa *container* memakan banyak sumber daya dalam hal penggunaan CPU, hasil untuk layanan *container Docker* adalah penggunaan CPU 84,45% dan penggunaan RAM 675 Mb. layanan *Podman* mendapatkan hasil penggunaan CPU 94,05% dan 658 Mb penggunaan RAM.

Penelitian [14] oleh Fernando Mardi Nurzaman, Ferdi Chahyadi, Muhamad Radzi Rathomi pada tahun 2022 yang berjudul "Analisis Perbandingan Performa *Load Balancer Nginx* Dan *Haproxy* Pada *Docker"* menjelaskan Implementasi cluster pada web *server* mengadopsi metode *load balancing*, sehingga beban trafik dari masing-masing web *server* dapat berjalan secara optimal. Kemudian aktifkan algoritma Round Robin untuk membagi setiap proses secara merata di antara *server* web. Dengan implementasi *Nginx* dan Haproxy *Load Balancer*, virtualisasi *container* dilakukan di *Docker*. Hasil penelitian mengungkapkan bahwa Haproxy *Load Balancer* pada *Docker* yang menerapkan Round Robin mengungguli *Load Balancer* lainnya. Rata-rata nilai *throughput* yang didapat dari uji performa adalah 8418.6 KB/s, nilai rata-rata

response time adalah 2.2 ms, nilai rata-rata penggunaan CPU adalah 42.0% dan nilai rata-rata penggunaan memori adalah 0.4% dari beban, Haproxy Balancer pada *Docker* dan lainnya Penyeimbang beban relatif stabil. Ketersediaan pada koneksi yang padat (kemacetan) dan permintaan yang besar dapat dilakukan tanpa galat menggunakan *Load Balancer* Haproxy di *Docker*, sedangkan pada *Load Balancer Nginx* di *Docker* mendapatkan nilai rata-rata galat sebesar 739.8 connection *time*outs.

Penelitian [15] oleh Muhammad Abdul Aziz, Adhitya Bhawiyuga, Fariz Andri Bakhtiar pada tahun 2020 yang berjudul "Implementasi *Container Live Migration* Antar-*Cloud Provider* Menggunakan *Podman* dan CRIU" Menggunakan *Podman* dan CRIU, teknik migrasi langsung *container* diimplementasikan untuk memindahkan layanan dari satu VM ke VM lainnya tanpa memindahkan semua status dari VM sumber ke VM target. *Podman* merupakan sebuah layanan yang dapat berjalan secara independen terlepas dari lingkungannya. CRIU dapat digunakan untuk menghentikan dan memindahkan satu mesin ke mesin lainnya. Hasil pengujian menjelaskan bahwa metode migrasi langsung yang digunakan dapat melakukan migrasi layanan antar penyedia *cloud* yang berbeda. Hasil penelitian yang penerapan *lift-and-shift container* antara penyedia *cloud* menggunakan *Podman* dan CRIU menunjukkan waktu henti rata-rata 34.618 detik dan waktu migrasi rata-rata 71.627 detik.

Pada penelitian yang dilakukan oleh Chairul Mukmin dkk pada tahun 2021 tentang studi kasus *Docker* dan *Podman*. Lalu penelitian yang dilakukan oleh Albert Yakobus Chandra pada tahun 2019 yang membahas tentang Apache *vs Nginx*. Dari kedua penelitian tersebut, maka alasan penulis dalam menggunakan judul dengan *Docker* dan *Podman* sebagai *container* serta menggunakan web *server* berupa *Nginx* adalah karena penelitian tersebut membahas hal yang akan dijadikan bahan pada penelitian kali ini.

Tabel 2. 1 Penelitian Sebelumnya

No	Judul	Penulis, Tahun	Metode	Hasil	Perbedaan Penelitian
1.	Analisis Performansi Antara Apache & *Nginx* Web *Server* dalam Menangani *Client Request* [1]	Albert Yakobus Chandra, 2019	*Benchmark Test*	Hasil pengujian dari penelitian ini memberikan hasil *benchmark* yang menunjukkan bahwa *Nginx* membutuhkan waktu lebih sedikit untuk merespon permintaan klien dibandingkan Apache dalam hal waktu penggunaan.	Penelitian ini hanya menggunakan *Nginx* sebagai web *server* serta *Docker* dan *Podman* sebagai *container*nya, sedangkan pada penelitian tersebut menggunakan Apache & *Nginx* sebagai web *server* serta tidak diketahui *container*nya.

No	Judul	Penulis, Tahun	Metode	Hasil	Perbedaan Penelitian
2.	Implementasi *Load balancing Server* Web Berbasis *Docker* Swarm Berdasarkan Penggunaan Sumber Daya *Memory Host* [16]	Mohamad Rexa Mei Bella, Mahendra Data, Widhi Yahya, 2019	*Load balancing, Failover*	Hasil penelitian menunjukkan bahwa *time-based failover* dan *memory* resource-based *Load Balancer* dapat bekerja di *Docker Swarm* yang dapat menangani *resource sharing* antar *host.*	Penelitian ini menggunakan *Docker* dan *Podman* sebagai *container*, sedangkan penelitian tersebut hanya menggunakan *Docker* sebagai *container*.
3.	Analisis Perbandingan *Server Load balancing* dengan Haproxy & *Nginx* dalam	Sampurna Dadi Riskiono dan Donaya Pasha, 2020	*Load balancing*	Hasil pengujian menunjukkan bahwa waktu respon penggunaan Haproxy untuk mencapai *load balancing* adalah 585ms, lebih rendah dari waktu respon *Nginx* pada pengujian koneksi 300/300 detik sebesar 1180.58ms. Untuk *load balancing*, nilai *throughput* Haproxy juga	Penelitian ini menggunakan benchmark *test*, sedangkan penelitian tersebut menggunakan *load balancing* sebagai metode perbandingan

No	Judul	Penulis, Tahun	Metode	Hasil	Perbedaan Penelitian
	Mendukung Kinerja *Server E- Learning* [3]			menunjukkan kinerja yang lebih baik, dengan nilai 896,48 Kb/s, lebih tinggi dari *Nginx* 848,52 Kbps pada uji koneksi 300/300 detik. Menurut pengujian yang dilakukan, implementasi *load balancing* Haproxy lebih baik daripada *Nginx*.	namun menggunakan web *server* yang sama.
4.	Implementasi *Container Live Migration Antar-Cloud Provider* Menggunakan *Podman* dan CRIU [15]	Muhammad Abdul Aziz, Adhitya Bhawiyuga, Fariz Andri Bakhtiar, 2020	*Live Migration*	Hasil pengujian fungsional menunjukkan bahwa metode migrasi langsung mampu melakukan migrasi layanan antar penyedia *cloud* yang berbeda, dan penyebaran *container* menggunakan *Podman* dan CRIU untuk migrasi langsung antar penyedia *cloud* menunjukkan waktu henti rata-rata 34.618 detik dan waktu migrasi rata-rata 71.627 detik.	Pada penelitian ini menggunakan *Docker* dan *Podman* sebagai *container*, sedangkan pada penelitian tersebut hanya menggunakan *Podman* sebagai *container*.

No	Judul	Penulis, Tahun	Metode	Hasil	Perbedaan Penelitian
5.	Analisa dan Implementasi *Microservice* pada *Container* Menggunakan *Docker* [8]	Stefanus Eko Prasetyo dan Ardyansyah Wijaya, 2021	*Load Testing*	Aplikasi Microservices di *Docker* mudah digunakan dan stabil saat menguji berbagai beban dengan Apache Jmeter.	Penelitian ini menggunakan *Docker* dan *Podman* sebagai *container*, sedangkan penelitian tersebut hanya menggunakan *Docker* sebagai *container*.
6.	Implementasi CDN (*Content Delivery Network*) Menggunakan *Cloud*flare Terintegrasi Dengan *Docker Container* [17]	Haikal Alham Tuara, NurAlif Maridyah, Khaerudin Khaerudin, 2021	CDN *(Content Delivery Network)*		Penelitian ini menggunakan *Docker* dan *Podman* sebagai *container*, sedangkan penelitian tersebut hanya menggunakan *Docker* sebagai *container*.

No	Judul	Penulis, Tahun	Metode	Hasil	Perbedaan Penelitian
7.	Analisis Perbandingan Kinerja Layanan *Container As A Service* (CAAS) Studi Kasus : *Docker* dan *Podman* [13]	Chairul Mukmin, Therezia Naraloka, Qodri Harun Andriyanto, 2021	*Benchmark Test*	Hasil menunjukkan bahwa *container* memakan banyak sumber daya dalam hal penggunaan CPU, hasil layanan *container Docker* adalah penggunaan CPU 84,45% dan penggunaan RAM 675 Mb, sedangkan hasil layanan *Podman* adalah penggunaan CPU 94,05% dan Penggunaan RAM 658 Mb Mb Penggunaan RAM Penggunaan memori.	Penelitian ini menggunakan *Docker* dan *Podman* sebagai *container* dan *Nginx* sebagai web *server*, perbedaanya adalah penelitian tersebut tidak diketahui menggunakan web *server* jenis apa namun menggunakan *container* yang sama.
8.	Implementasi *Load balancing* dan *Failover* pada Proses Migrasi *Container Docker* [5]	Shofura Naufal Rifiera dan Heru Nurwasito, 2022	*Load balancing, Failover*	*Load balancing* dan *failover* pada *Docker* sangat berguna untuk mendistribusikan layanan web *server* dari *node* manajemen ke *node* pekerja sebagai *server* cadangan jika terjadi kegagalan sistem salah satu *server*. Web *server* dinamis kemudian memiliki waktu respons rata-rata terendah	Penelitian ini menggunakan *Docker* dan *Podman* sebagai *container*, sedangkan penelitian tersebut menggunakan *Docker* sebagai *container*.

No	Judul	Penulis, Tahun	Metode	Hasil	Perbedaan Penelitian
				ke *node* pekerja 2 dengan nilai 18,90 ms, dengan waktu pengambilan dipengaruhi oleh durasi latensi yang ditentukan selama uji beban layanan *server* web.	
9.	Membuat Web *Server* Menggunakan Debian 10 Pada *Virtual Machine* [18]	Mahesadaru Wicaksono dan Johan Pamungkas, 2022	*Benchmark Test*	Hasil pengujian performa menggunakan tool Apache Benchmark menggunakan tiga skenario yaitu mengakses halaman admin, halaman awal, dan halaman sampel di Wordpress, dan masing-masing file diunggah yaitu 296 byte, 287 byte, dan 298 bagian kata-kata, waktu respons tidak boleh terlalu besar. Jumlah koneksi di setiap skenario adalah 1000, dan tingkat permintaan meningkat sebesar 10, 20, 30, 40, 50, 60, 70, 80, 90, dan 100. Dengan meningkatnya tingkat permintaan, *throughput* yang dihasilkan di setiap	Penelitian ini menggunakan Rocky Linux sebagai sistem operasi, sedangkan penelitian tersebut menggunakan Debian 10 sebagai sistem operasi.

No	Judul	Penulis, Tahun	Metode	Hasil	Perbedaan Penelitian
				skenario terus meningkat. Oleh karena itu dapat disimpulkan bahwa dengan meningkatkan nilai *throughput* maka Web dianggap cukup baik.	
10.	Analisis Perbandingan Performa *Load Balancer Nginx* Dan Haproxy Pada *Docker* [14]	Fernando Mardi Nurzaman, Ferdi Chahyadi, Muhamad Radzi Rathomi, pada 2022	*Load balancing*	Hasil pengujian menunjukkan bahwa Haproxy *Load Balancer* pada *Docker* yang menggunakan algoritma Round Robin mengungguli *Load Balancer* lainnya. Rata-rata nilai *throughput* yang didapat dari uji performa adalah 8418.6 KB/s, rata-rata *response time* 2.2 ms, rata-rata penggunaan CPU 42.0%, rata-rata penggunaan memori 0.4% dari beban, Haproxy Balancer pada *Docker* dan lainnya Penyeimbang beban relatif stabil. Ketersediaan pada koneksi yang padat (kemacetan) dan permintaan yang besar	Penelitian ini menggunakan benchmark *test*, sedangkan penelitian tersebut menggunakan *load balancing* sebagai metode perbandingan namun menggunakan web *server* yang sama.

No	Judul	Penulis, Tahun	Metode	Hasil	Perbedaan Penelitian
				dapat dilakukan tanpa error menggunakan *Load Balancer* Haproxy di *Docker*, sedangkan pada *Load Balancer Nginx* di *Docker* rata-rata error adalah 739.8 *connection timeouts*.	

2.2 Landasan Teori

2.2.1 VirtualBox

VirtualBox merupakan perangkat lunak yang dapat membuat, mengkonfigurasi, dan mengelola mesin virtual untuk melakukan sistem operasi virtual pada sistem operasi utama (*host*)[19].

VirtualBox juga dapat digunakan untuk membuat virtualisasi sederhana dari jaringan komputer. Penggunaan VirtualBox ditujukan untuk penggunaan *server, desktop* dan *embedded*. Berdasarkan tipe *virtual machine monitor* (VMM) yang ada, VirtualBox adalah *hypervisor type* 2[20].

2.2.2 Web *Server*

Web *server* adalah perangkat lunak integrasi *server* yang menerima permintaan (*request*) untuk situs web dari program browser web melalui HTTP atau HTTPS dan merespons dengan halaman web, biasanya dalam bentuk dokumen HTML[21].

Keberadaan web *server* memudahkan pertukaran informasi antara klien dan *server* dalam jaringan. Saat membuat *server* web, diperlukan beberapa konfigurasi alamat IP, DHCP, DNS, dan basis data[18]. Namun, konfigurasi pemrograman web *server* tidak diperlukan pada *website* statis[22]. Beberapa contoh web *server* yang dapat digunakan antara lain, *Nginx,* Apache, Hiawatha, Cherokee, Lighttpd dan Apache Tomcat[23].

2.2.3 *Nginx*

Server HTTP *open source Nginx* memiliki performa yang cepat dan efisien. *Nginx* pertama kali dikembangkan untuk mengatasi masalah C10K yang muncul saat 10.000 komputer beroperasi secara bersamaan. *Nginx* dapat menggunakan sumber daya secara efektif karena desainnya yang digerakkan oleh peristiwa (asinkron), yang memungkinkan

penghematan daya yang lebih besar. *Nginx* juga menawarkan serangkaian fitur dan mudah dikonfigurasi[24].

2.2.4 *Container*

Container merupakan sebuah teknologi alternatif yang dapat digunakan untuk virtualisasi sistem komputer. Dengan menggunakan *Container,* dapat mengemas program komputer (kode) ke dalam unit standar sehingga semua dependensi aplikasi dapat disertakan dalam paket ini, seperti kode, *runtime,* alat sistem, pustaka sistem, dan pengaturan. *Container* memungkinkan aplikasi dipindahkan dengan cepat dari satu lingkungan desktop ke lingkungan lainnya. Paket *container* mencakup semua yang dibutuhkan untuk menjalankan aplikasi[25].

2.2.5 *Docker*

Docker adalah solusi perangkat lunak yang bertindak sebagai sebuah wadah untuk menjalankan aplikasi dengan menerapkan teknik *container* untuk memisahkan setiap layanan dan lingkungannya[26].

Docker bekerja dengan menyediakan cara standar untuk menjalankan perintah pengguna. *Docker* berfungsi untuk wadah. Dengan cara yang sama seperti mesin virtual melakukan virtualisasi (menghilangkan kebutuhan untuk mengelola secara langsung) perangkat keras *server*, *container* memvirtualisasikan sistem operasi *host*. *Docker* diinstal pada setiap *host* dan menyediakan perintah sederhana yang dapat digunakan untuk membuat, memulai, atau menghentikan kontainer[27].

2.2.6 *Podman*

Podman merupakan *container* run*time* yang kompatibel dengan OCI yang bekerja tanpa daemon dan CLI mengeksekusi semua perintah inti *Docker*. Dalam penggunaanya, mudah untuk beralih ke *Podman* atau

menggunakannya bersama dengan instalasi *Docker* yang ada. Tidak seperti *Docker*, *Podman* memberikan dukungan terbaik untuk mengelola banyak *container*. Model Pod memudahkan untuk bekerja dengan banyak layanan. *User* dapat menghentikan, memulai ulang, dan menghapus semua *container* terkait menggunakan perintah tingkat pod[11].

2.2.7 Apache Benchmark

Apache Benchmark adalah sebuah alat yang dapat mengukur kinerja web *server*. Menguji kemampuan web *server* untuk memenuhi kebutuhan *client*. Parameter yang terdapat pada Apache benchmark dalam perintah yang digunakan antara lain, *concurrency* atau -c = menyatakan jumlah *request* akses yang dilakukan oleh user dalam waktu bersamaan. Sedangkan *request* atau -n = menyatakan jumlah koneksi yang dibuat ke *server* tujuan[28].

Beberapa fitur dari Apache Bench seperti: *open source, simple command line, platform independent, load and performance test, not extensible*. Alat ini dapat digunakan untuk menguji kinerja *server* web dengan berbagai batasan pengujian seperti response *time* yang berisi permintaan per-detik dan waktu per-permintaan[23].

2.2.8 *Response Time*

Parameter dalam hasil pengujian *response time* terbagi menjadi tiga bagian, yaitu *Request* per-*Second, Time* per-*Request*, *Time* per-*Request (concurrent)*[29].

Request per-*Second* merupakan ukuran jumlah permintaan yang diterima oleh sistem atau *server* dalam satu detik. RPS digunakan untuk mengukur kinerja dan skalabilitas sistem sesuai dengan kebutuhan pengguna[30]. Lalu *Time* per-*Request*, merupakan ukuran waktu rata-rata yang dibutuhkan sistem untuk menanggapi permintaan dari pengguna. Semakin rendah waktu per permintaan, semakin cepat sistem

merespons, dan ini merupakan indikasi kinerja yang baik[23]. Sedangkan *Time* per-*Request (concurrent*) merupakan ukuran waktu rata-rata yang diperlukan untuk menanggapi permintaan saat beberapa permintaan sedang diproses secara bersamaan[31].

2.2.9 Prometheus

Prometheus merupakan *software* yang berbasis *open-source*, digunakan sebagai alat *monitoring* dan *alerting* pada sebuah sistem[6]. Salah satu keunggulan Prometheus dibandingkan lainnya adalah memiliki banyak metrik (pengukuran) yang diperlukan untuk memantau sistem. Prometheus mudah disesuaikan, membuatnya sangat berguna saat berintegrasi dengan perangkat lunak lainnya. seperti saat digunakan dengan Grafana untuk kebutuhan visualisasinya[32].

2.2.10 Grafana

Grafana adalah *software* visualisasi dan analisis berbasis *open source*. Grafana memungkinkan melihat, memberi peringatan, dan menjelajahi metrik yang disimpan. dan mengubah data dari *Time Series Database* (TSDB) menjadi visualisasi yang indah[33]. Seperti dari matriks yang diperoleh dari perangkat lunak seperti Prometheus.