BAB II
DASAR TEORI

2.1 KAJIAN PUSTAKA

Penelitian yang dilakukan oleh Maya Rosalia dkk dimana penerapan *cluster* dilakukan menggunakan metode virtualisasi, didapat data berupa *throughput* maksimal pada 1.000 *request* sebesar 0,846 Mb/s, bernilai 2,092 Mb/s pada 3.000 *request*, 2,702 Mb/s pada 7.000 *request* dan pada 10.000 *request* sebesar 3,53 Mb/s sedangkan untuk parameter *cpu utilization* rata-rata vm pada 1.000 *request* bernilai 10%, pada 3.000 *request* sebesar 20%, saat 5.000 *request* bernilai 38%, lalu bernilai 40% pada 7.000 *request* dan bernilai 50% pada saat 10.000 *request*[8].

Penelitian yang dilakukan oleh Jafaruddin Gusti Amri Ginting, dkk tahun 2021 melakukan analisis dan implementasi teknik failover menerapkan load balancing menggunakan Google Kubernetes Engine (GKE) yang diterapkan pada *cloud environment*. Dari penelitian tersebut dihasilkan rata-rata *availability* sebesear 99,96% serta rata-rata *throughput* pada 200 koneksi sebesar 5,599 Mb/s, pada 500 konseksi mengalami peningkatan menjadi 6,216 Mb/s dan pada 1000 koneksi bernilai 8,764 Mb/s, sedangkan pada 2.000 koneksi mengalami penurunan *througput* menjadi 7,581 Mb/s dan pada 5.000 koneksi bernilai 7,473 Mb/s. Selain itu parameter *resource utilization* menghasilkan data pada 200 koneksi, CPU *ussage* sebesar 9,597%, lalu 15,002% pada 500 koneksi, 19,411% pada 1.000 koneksi, 24,021% pada 2.000 koneksi, dan pada 5.000 koneksi CPU *ussage* sebesar 27,178%[9].

Penelitian yang dilakukan oleh Kholifah Dyah Ayatri yang mana melakukan implementasi dan analisis terhadap *High Availability Cluster Server* Menggunakan Heartbeat yang diterapkan pada *private cloud environment*, didapat data berupa parameter *Availability* bernilai rata-rata 99,95%. Untuk parameter *Delay* bernilai 1,43 detik, 1,82 detik, 1,65 detik, 1,73 detik, 1,70 detik pada pengujian dengan 200, 400, 600, 800, dan 1.000 koneksi secara berurutan. Sedangkan pada parameter *Throughput* memiliki nilai 2,652 Mb/s pada 200 koneksi, 5,435 Mb/s pada 400 koneksi, 6,244 pada 600 koneksi, 6,144 Mb/s pada 800 koneksi dan pada 1.000 koneksi memiliki nilai *throughput* sebesar 5,807 Mb/s.[10]

Penelitian dari Chaerul Umam dkk berjudul "*Implementation And Analysis High Availability Network File System Based Server Cluster*" yang mana melakukan implementasi dan melakukan analisa *high availability* cluster difungsikan untuk melayani *Network File System*. Dari penelitian tersebut didapat hasil *throughput* 20MB/s dan *availability* sebesar 99,998%[11].

Penelitian yang dilakukan oleh Muhammad Aldi Aditia Putra dkk, melakukan implementasi *high availability cluster* web *server* menggunakan virtualisasi container docker yang disertai menggunakan *load balancer external* menggunakan HAProxy pada 2 mesin docker yaitu 1 manager dan 1 *node* menghasilkan data berupa *Throughput* tertinggi ketika *server* menerima 7.000 koneksi sebesar 9,22 MB/s, pada parameter *Delay* saat *server* medapat 7.000 koneksi saat menggunakan algoritma Round Robin memiliki *Delay* sebesar 2,128 detik, dan 2,127 detik saat diterapkan algoritma *least connections*[12].

Penelitian yang dilakukan oleh Muhammad Yusuf Maulana tahun 2021 melakukan percobaan pada *virtual environtment* menerapkan docker swarm sebagai metode *load balancing* dimana terdapat 3 *server* virtual salah satunya menjadi *manager* dan dua diantaranya menjadi *worker node*. Melakukan analisa parameter *availability* serta *cpu utilization*[13].

Penelitian yang dilakukan oleh T. Yudi Hadiwandra dkk, melakukan implementasi docker swarm pada raspberry pi *cluster* yang berisikan 5 buah raspberry pi 3 diantaranya adalah *manager*, dan 2 yang lain merupakan *worker node*, sehingga dapat dicapai 2 *fault tolerance*. Didapatkan hasil throughput maksimal pada 20.000 koneksi memiliki kecepatan 161.812.298 bytes/s atau sebesar 150.316 MB/s[14].

Tabel 2.1 Perbandingan dengan penelitian sebelumnya

Author	*Deployment*	Teknik Failover	Perangkat Failover
Maya Rosalia, dkk	Virtual Cluster	Load Balance	Nginx
Jafaruddin Gusti Amri Ginting, dkk	Cloud	Load Balance	GKE
Kholifah Dyah Ayatri	Cloud	Heartbeat	Openstack
Chaerul Umam, dkk	Local Physical Cluster	Load Balance	Pacemaker
Muhammad Yusuf Maulana	Virtual Cluster	Load Balance	Docker Swarm
T. Yudi Hadiwandra, dkk	Pi-Cluster	Load Balance	Docker Swarm
Rosyid U. Ma'ruf	Pi-Cluster	Load Balance	Docker Swarm

2.2 DASAR TEORI

2.2.1 Load balancing

Load Balancing merupakan sebuah teknik untuk membagi beban *traffic* dan membaca ketersedian *resource* dari sebuah *server*[15]. Fungsi ini digunakan untuk mengurangi beban berlebih pada salah satu *server* dari beberapa *server* yang tersedia, sehingga beban pemrosesan dapat disama ratakan pada tiap-tiap *server* yang tersedia.

2.2.2 Failover

Failover adalah suatu teknik untuk melakukan *switching* atau perpindahan dari sebuah *server* yang mengalami kegagalan menuju *server standby* yang berfungsi sebagai *hot-spare* untuk menggantikan *server* utama apabila terjadi kegagalan [16]. Hal ini bertujuan untuk mencapai ketersediaaan akses dari suatu layanan atau aplikasi dapat tetap diakses, meskipun terjadi kegagalan pada salah satu *server*.

2.2.3 Teknologi Virtualisasi

2.2.2.1 Vmware Vsphere

Vmware Vsphere merupakan sebuah *computing environment* untuk menjalankan *virtual machine*, container, dan kubernetes. Vsphere memungkinkan administrator untuk me-*manage* aplikasi modern yang kompleks semudah menjalankan aplikasi atau VM (*Virtual Machine*) tradisional. Arsitektur Vsphere juga mendukung pengembangan aplikasi berbasis container[17].

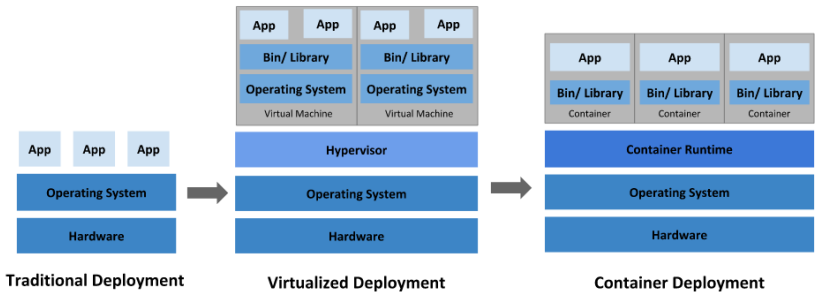
Use Case yang dapat dilakukan dengan Vsphere adalah diantaranya :

a. *Artificial Intelligence & Machine Learning*
b. Kegunaan untuk cabang dan kantor jarak jauh.
c. *Big Data* & Aplikasi data modern.
d. *High Performance Computing* (HPC).

2.2.2.2 Kubernetes

Kubernetes merupakan suatu platform portable untuk me-*manage container* dari layanan dan alur kerja, yang membolehkan administrator untuk melakukan konfigurasi secara deklaratif dan automatisasi. Selain banyaknya fitur, kubernetes juga memiliki ekosistem yang berkembang sangat pesat serta dukungan, layanan dan *tools* yang banyak tersedia[18].

Beranjak dari teknologi tradisional yang melakukan *deployment* pada satu mesin fisik yang khusus untuk menjalankan aplikasi/layanan tertentu, berkembang menuju virtualisasi, yang akhirnya berubah menjadi kontainerisasi seperti pada gambar 2.1.



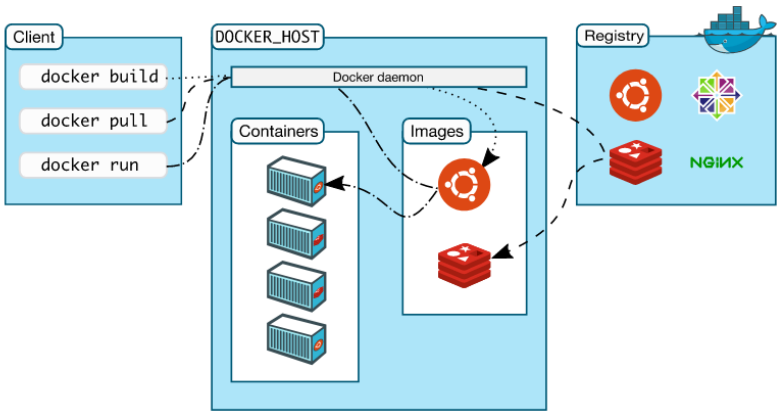
Gambar 2. 1 Perubahan mode *deployment*[18]

2.2.2.3 Docker

Docker merupakan salah satu platform *open source* yang digunakan untuk melakukan virtualisasi, pengembangan, dan menjalankan atau *deployment* dari suatu aplikasi[19]. Docker menyediakan kemampuan untuk melakukan isolasi atau pembatasan terhadap *environment* yang mana disebut sebagai *container*. Dengan dilakukannya isolasi pada tiap-tiap container, maka dapat dicapai pembagian *resource* yang maksimal, sehingga suatu aplikasi tidak menggunakan terlalu banyak *resource* yang mana sebenarnya tidak diperlukan. Selain itu docker *container* juga memudahkan pengembang untuk mengirimkan hasil aplikasinya kepada pengembang lain, maupun menjalankannya di lingkungan produksi, karena sifat *container* yang terisolasi, sehingga perubahan lingkungan luar tidak akan berpengaruh terhadap kinerja *container* yang dipindahkan.

A. Docker *Architecture*

Arsitektur docker menggunakan metode *client-server* dimana *client* docker berkomunikasi kepada docker daemon yang bertugas untuk *building*, menjalankan, dan mendistribusikan docker *container*[19]. Gambar 2.2 merupakan ilustrasi arsitektur docker yang hanya mengambil modul-modul yang diperlukan saja, sehingga dapat meringkas ukuran.



Gambar 2.2 Arsitektur Docker [19]

B. Docker *Images*

Merupakan sebuah *read-only template* dimana di dalamnya terdapat instruksi untuk membuat docker *container*[19]. Sebuah docker image dibangun

berdasarkan dari aplikasi yang dikembangkan oleh *developer* yang mana dalam docker *image* tersebut hanya terkandung modul-modul yang dibutuhkan oleh aplikasi itu saja, dan tidak memuat modul lain.

C. Docker Swarm

Docker swarm adalah sebuah fitur yang secara *default* telah ada di docker Engine yang mana membolehkan pengembang untuk melakukan *cluster management* dan *orchestration*[5]. Dengan menggunakan docker swarm, pengembang dapat dengan mudah untuk melakukan melakukan manajemen dari sebuah aplikasi yang hendak atau sudah di*deploy*. Selain itu docker swarm juga membolehkan aplikasi untuk dijalankan secara tidak tersentral atau *distributed*, sehingga dapat dilakukan *load balancing* dan *failover*.

Pada konfigurasi swarm mode, administrator harus memperhatikan jumlah *quorum* (jumlah anggota minimal) yang ada pada suatu *cluster*, agar pada saat terjadi kegagalan. Jumlah anggota minimal disini merupakan jumlah minimal perangkat yang aktif, sehingga *cluster* tidak kehilangan kemampuan untuk melakukan manajemen *cluster*.

cluster tersebut memiliki *fault tolerance* atau toleransi terhadap suatu kegagalan perangkat yang mana banyaknya toleransi yang dimiliki, bergantung pada jumlah *manager* dan *worker node* yang ada pada *cluster* tersebut.

Docker swarm menggunakan algoritma Raft Consensus untuk melakukan pengaturan *global cluster state*, dimana untuk mencapai *state* atau keadaan yang sama pada seluruh *cluster* terkait siapa yang menjadi manager pada *cluster* tersebut. Untuk menghitung berapa banyak *quorum* dan *fault tolerance* dari *cluster* docker swarm dapat dihitung menggunakan persamaan 2.1 dan 2.2 :

$$quorum = \frac{N}{2} + 1 \tag{2.1}$$

$$Fault\ Tollerance = \frac{N-1}{2} \tag{2.2}$$

Dimana N merupakan banyaknya perangkat yang di*cluster* / *Swarm Size* , misalkan suatu *cluster* docker swarm terdapat 3 *node*, menurut persamaan 2.1, maka diperlukan *quorum* sebanyak 2 *node*, dan apabila *server* yang mengalami kegagalan lebih dari batas *quorum* maka layanan tidak akan dapat diakses[19].

Berikut merupakan tabel petunjuk yang diberikan oleh dokumentasi dari docker dengan mengacu pada penerapan ketentuan yang diberikan oleh algoritma Raft Consensus:

Tabel 2. 2 Relasi ukuran *cluster* dengan minimal quorum dan *fault tolerance*[19]

Swarm *Size*	*Majority*	*Fault Tolerance*
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3
8	5	3
9	5	4

Suatu node dikatakan beroperasi sebagai manager *node* yang aktif apabila ketika dilakukan pengecekan menggunakan perintah *"docker node ls"* menunjukan bahwa pada "*Manager Status" node* tersebut terlabeli sebagai *"Leader"*, untuk manager yang beroperasi sebagai redundan atau *hot-spare* akan terlabeli sebagai "*Reachable*", sedangkan *node* yang beroperasi sebagai *worker-node* pada tabel "*Manager Status*" *node* tersebut tidak memiliki label.

2.2.4 Mini PC

Mini PC atau *minicomputers* adalah varian komputer yang mampu memproses data dan fitur seperti halnya komputer pc pada umumnya akan tetapi memiliki ukuran fisik yang kecil[20].

2.2.3.1 Orange Pi

Orange Pi adalah sebuah platform *Open Source Hardware* dan *Software* yang diperuntukkan untuk para antusias dan *creator* atau *developer* agar dapat menikmati perangkat yang berkualitas tinggi dengan harga terjangkau.

Menawarkan perangkat *Single Board Computer* yang memiliki paket lengkap dan terjangkau[21].

Gambar 2. 3 Orange Pi 3 LTS[21]

2.2.3.2 NVIDIA Jetson

Nvidia Jetson merupakan *minicomputer* yang digunakan sebagai platform untuk *autonomous* machines dan *embeded system* lainnya. Setiap NVIDIA Jetson adalah *System on Module* lengkap yang didalamnya terdapat *Graphic Processing Unit* (GPU), *Central Processing Unit* (CPU), memori, *Power Management¸Interface* dan hal-hal lain yang mendukung kerja *minicomputer*[22].

Gambar 2. 4 Nvidia Jetson Nano

2.2.3.3 Raspberry Pi

Raspberry Pi merupakan sebuah komputer yang terjangkau dan portabel, perangkat ini membolehkan siapa saja untuk mendalami berbagai hal tentang *computing*[23]. Dengan menggunakan raspberry pi, dapat dicapai perancangan *cluster* dengan biaya dan konsumsi daya yang relatif kecil.

Gambar 2. 5 Raspberry Pi 3 Model B[23]

2.2.5 Benchmarking Tools

2.2.4.1 Locust

Locust merupakan *performance testing tool software* yang mudah digunakan, *scriptable* dan *scalable*[24]. Pada locust penguji dapat melakukan kustomasi pengujian dengan mengubah ataupun menambahkan *custom script* pada program, karena pada dasarnya locust merupakan perangkat lunak berbasis pada bahasa pemrograman python.

2.2.4.2 Apache Jmeter

Berbeda dengan Locust yang berbasis pada python, Apache Jmeter merupakan *software* yang 100% berbasis pada desain aplikasi Java yang digunakan untuk melakukan *load test functional behavior* dan mengukur performansi dari aplikasi web[25].

2.2.4.3 Siege

Siege merupakan *open source software* yang digunakan untuk melakukan *regression test* dan *benchmark utility*. *Software* ini dapat melakukan *stress test* pada sebuah URL dengan menggunakan *user defined number* sebagai simulasi *user* melakukan akses pada URL tersebut, *software* ini juga dapat membaca banyak URL dan melakukan *stress test* secara bersamaan. Dengan menggunakan sege, dapat dikonfigurasikan agar terdapat *concurrent user* atau simulasi pengaksesan *user* secara bersamaan, sehingga dapat diketahui batas maksimal yang dapat diproses oleh cluster[26].

2.2.6 Monitoring Tools

2.2.5.1 Prometheous

Prometheous merupakan *open-source software* yang digunakan untuk melakukan monitoring dan *alerting*[27]. Prometheus mengumpulkan matriks yang didapat dari sistem yang sedang dilakukan monitor dan menyimpannya bersama dengan stampel waktu data tersebut.

2.2.5.2 Grafana

Grafana merupakan *multi-platform open-source software* yang membolehkan pengguna untuk melakukan visualisasi dari matriks yang didapat dari *software* seperti prometheus, sehingga dapat dengan mudah dipahami. Grafana juga menyediakan fitur *alerting* jika terdapat suatu *event* dimana administrator harus diberitahu.

2.2.7 Parameter Pengujian

2.2.6.1 High Availability

Di dunia Komputasi kata *availability* digunakan untuk menjelaskan periode waktu dari ketersediaan sebuah layanan[28]. Infrastruktur dengan kemampuan *High Availability* dirancang untuk menyediakan kualitas, performa dan dapat menangani berbagai macam *load* serta kegagalan perangkat dengan *downtime* yang minimal atau mendekati nol.

2.2.6.2 Failover

Failover merupakan metode operasional yang secara otomatis melakukan pemindahan jalur dari *resource* yang mengalami kegagalan menuju *resource* yang *standby*[29]. Dengan menggunakan metode *Failover* dapat dicapai suatu layanan yang memiliki *High Availability.*

2.2.6.3 Availability

Availability merupakan suatu prosentase suatu sistem tersedia atau *available*[30]. Nilai ketersediaan dari suatu sistem dapat dihitung menggunakan persamaan 2.3 :

$$availability = \frac{uptime}{uptime+downtime} \quad (2.3)$$

Dimana *downtime* merupakan keadaan suatu sistem tersebut tidak tersedia atau *down* baik itu diharapkan/dijadwalkan maupun terjadi diluar perkiraan. Selain menggunakan rumus diatas, nilai *availability* juga dapat didapat menggunakan persamaan berikut :

$$availability = \frac{MTBF}{MTBF+MTTR} \quad (2.4)$$

MTBF (*mean time between failures*) yaitu jarak atau rentang waktu dari sistem mengalami kegagalan, atau dengan kata lain waktu suatu sistem beroperasi dengan normal dan MTTR (*Mean Time To Repair*) adalah rata-rata waktu yang dibutuhkan untuk melakukan *troubleshooting* sehingga suatu sistem yang mengalami kendala dapat beroperasi dengan normal kembali [30].

Kategori parameter availability dilihat dari "nines", maksudnya adalah tingkat prosentase ketersediaan yang mencapai 99,99 persen. Dimana semakin banyak angka sembilan dibelakang koma, maka kategori tingkat ketersediaan sistem tersebut semakin baik[31]. Menurut standar TYPHON TS 102 024-12, nilai *availability* yang baik dari suatu layanan adalah diata 99,99% sehingga apabila nilai *availability* sebesar 99,998% maka dalam satu tahun, layanan tersebut memiliki *down time* selama 88 jam per tahun.

2.2.6.4 Throughput

Throughput merupakan suatu ukuran dari nilai rata-rata data yang mampu dikirimkan suatu perangkat atau sistem dalam rentang waktu tertentu, memiliki satuan bit/s. Nilai *throughput* dapat dihitung menggunakan persamaan 2.5 [32].

$$Throughput = \frac{Data\ yang\ dikirim\ (bit)}{Durasi\ pengiriman\ data\ (second)} \quad (2.5)$$

2.2.6.5 Response Time

Response Time adalah waktu yang dibutuhkan suatu sistem untuk mengirimkan data yang diminta oleh *client*. Atau pada *web server, response time* adalah seberapa lama waktu yang dibutuhkan oleh *server* untuk memuat aset yang diperlukan untuk menampilkan halaman yang diminta oleh *client*[33].

2.2.6.6 Delay

Delay merupakan jeda waktu yang dibutuhkan oleh suatu paket untuk dikirimkan dari sumber hingga sampai ketujuannya (baik dari sisi *client* maupun *server*)[34].

Tabel 2. 3 Kategori Delay Menurut Standarisasi

Kategori	*Delay*
Best	< 150 ms
High	151 ms - 250 ms
Medium	251 ms - 350 ms
Low	351 ms - 450 ms
Tidak Direkomendasikan	> 450 ms