BAB 2
DASAR TEORI

2.1 Kajian Pustaka

Penelitian Suryanto dkk. [7] pada tahun 2018, menganalisis performansi mekanisme akses *e-learning* berbasis *linux container* pada komputer lokal. Penelitian ini bertujuan untuk membandingkan sistem lama dengan sistem baru. Perbedaanya adalah sistem lama tidak menggunakan *reverse proxy* sedangkan sistem baru menggunakannya. Dalam penelitian tersebut, dilakukan skenario pengujian dengan jumlah user 100, 250, dan 500 dengan 1 kali dan 20 kali akses dalam 1 menit. Parameter uji yang diukur adalah CPU, memori, *latency* dan *throughput*. Dari hasil pengujian didapatkan bahwa performa dari penggunaan CPU di beberapa skenario sistem lama lebih baik dari pada sistem baru. Hal tersebut dapat dipengaruhi karena pada sistem baru terdapat beberapa aplikasi tambahan seperti *nginx* dan *bind9*. Namun dari segi penggunaan memori kedua sistem tidak terdapat perbedaan yang signifikan. Dari hasil pengujian dapat dilihat bahwa waktu respon, baik *latency* maupun *troughput* di beberapa skenario tidak memiliki perbedaan yang signifikan. Namun ketika terdapat user yang mengakses berjumlah banyak (dalam skenario terakhir) *throughput* yang dihasilkan pada sistem baru lebih baik daripada sistem lama.

Penelitian Abdillah [8] pada tahun 2022, menganalisa performansi *web server* menggunakan *load balancing* pada virtualisasi *docker container*. Penelitian ini bertujuan untuk menguji seberapa besar pengaruh dari performansi *web server* dengan menggunakan *load balancing* yang dilengkapi oleh *docker container* pada komputer lokal. Dimana pada penelitian ini pengujian dilakukan dengan mengirim beberapa *request* ke *web server* dengan dua skenario, yaitu *load balancing web server* 2 *node* dan 3 *node*. Pengujian dilakukan dengan mengirimkan *request* secara bertahap yaitu, 50 *request* dan 25 *concurent*, 100 *request* dan 50 *concurent*, 200 *request* dan 1000 *concurent*, 400 *request* dan 200 *concurent*, 800 *request* dan 400 *concurent* menggunakan *apache benchmark*. Parameter yang digunakan ialah *taken for test*, *Request Per Second* (RPS), dan *Time Per Request* (TPR). Didapatkan hasil bahwa dalam pengujian terhadap *load balancing* dengan

menggunakan 2 *node* dan juga *load balancing* menggunakan 3 *node*, menghasilkan waktu dalam melakukan tes dengan 2 *node* yang lebih cepat daripada 3 *node*. Untuk *request per second*, *load balancing* dengan 3 node memiliki waktu *request* per detik lebih kecil daripada 2 *node*. Kemudian time per *request* yang dibutuhkan untuk *load balancing* 2 *node* lebih sedikit.

Penelitian Riskiono dkk. [9] pada tahun 2020, menganalisa metode *load balancing* dalam meningkatkan kinerja *website e-learning*. Penelitian ini bertujuan untuk mengetahui kinerja *website e-learning* dengan membandingkan *server* tunggal dengan *server* jamak dan mebandingkan algoritma *round robin* dengan *least connection* yang diimplementasikan pada *server* jamak. Dimana pada penelitian ini pengujian dilakukan menggunakan 3 skenario dengan melihat nilai *respone time* dan nilai *throughput*. Pengujian pertama dilakukan untuk mengetahui nilai *respone time* pada *server* tunggal dan *server* jamak dengan hasil *server* jamak menunjukkan bahwa kecepatan *respone time* yang didukung *load balancing* lebih cepat dibandingkan dengan *server* tunggal. Pengujian selanjutnya dengan membandingkan algoritma *round robin* dan *algoritma least connection* pada *server* jamak dengan rata-rata hasil menunjukkan *least connections* memiliki nilai *response time* 50.27 ms lebih kecil dibandingkan algoritma *round robin* yang memiliki waktu respons 50.88 ms. Ini menunjukkan respon algoritma *least connections* lebih cepat dibanding algoritma *round robin*. Untuk *throughput*, algoritma *least connection* juga memiliki nilai yang lebih besar yaitu 29.51 Kb/*sec* dibandingkan algoritma *round robin* yang memiliki nilai 29.21 Kb/*sec*. Ini menunjukkan bahwa jumlah paket yang dapat dilewatkan *least connection* lebih besar dari pada *round robin.*

Penelitian Sofyan dkk. [10] pada tahun 2022, menganalisa *load balancing web server* menggunakan *haproxy* pada virtual *server* direktorat SMK Kemendikbudristek. Penelitian ini bertujuan untuk menerapkan sistem *load balancing* menggunakan *haproxy* pada virtualisasi infrastruktur dan membuat *server backup* dengan sistem *load balancing* menggunakan *haproxy* dengan metode algoritma *round-robin*. Pengujian di lakukan dengan cara *stress test* menggunakan aplikasi *Jmeter* untuk mengukur performa suatu *web*. Dimana menggunakan perhitungan *response time testing* dengan meneliti nilai *Sample*

Time dan status yang telah didapatkan dari *Apache Jmeter*, dan perhitungan diskemakan dengan sampel 1000 *user* dalam 10 detik dan 1 kali pengetesan. Hasil pengetesan pada menu *summary report* perhitungan *respone time testing* dengan meneliti nilai *Min*. dan *Max*. dihasilkan respon *Min*. 13 dan *Max*. 81 yang berarti kecepatan respon *website* masih normal dengan sample 893 *user* yang mengakses.

Penelitian Apriliansyah dkk. [11] pada tahun 2020, menganalisa *load balancing* pada *web server* menggunakan *nginx*. Penelitian ini bertujuan untuk membandingnkan algoritma *round robin*, *least connection*, dan IP *hash*. Dimana pada penelitian ini menggunakan 7 computer yang berperan sebagai klien dan *server*. Dari 7 computer tersebut, 1 diantaranya berperan sebagai load balancing, 6 sebagai *server*. Pengujian dilakukan dengan memberikan *request* secara bertahap yaitu, 4000, 8000, 12000, 16000 *request*. Kemudian melakukan perbandingan dengan menggunakan parameter *throughput*, *respone time*, *request per second*, *downtime*, dan CPU *Utilization*. Didapatkan hasil terbaik adalah algoritma *least connection* karena mempunyai stabilitas *response time* 116ms yang signifikan dan kecepatan *response time* nya lebih bagus dengan mendapatkan nilai 2.300,96 req/s dan *throughput* 17,380,01 kbps. Berikut Tabel 2. 1 mengenai rangkuman keterkaitan dengan penelitian sebelumnya.

Tabel 2. 1 Rangkuman Keterkaitan dengan Penelitian Sebelumnya

Penelitian Oleh	Parameter Penelitian				
	Objek	Metode	Resource	Database	Parameter
Suryanto [7]	E-learning	Docker container	lokal	Mysql	CPU, memori, latency, dan throughput
Abdillah [8]	Web server	Docker container dan load balancing	Lokal	Mysql	taken for test, Request Per Second (RPS), dan Time Per Request (TPR)
Riskiono dkk. [9]	E-learning	Load balancing	Lokal	Mysql	Throughput dan respon time
Sofyan dkk.	Web server	Load	Cloud	Mysql	Respone

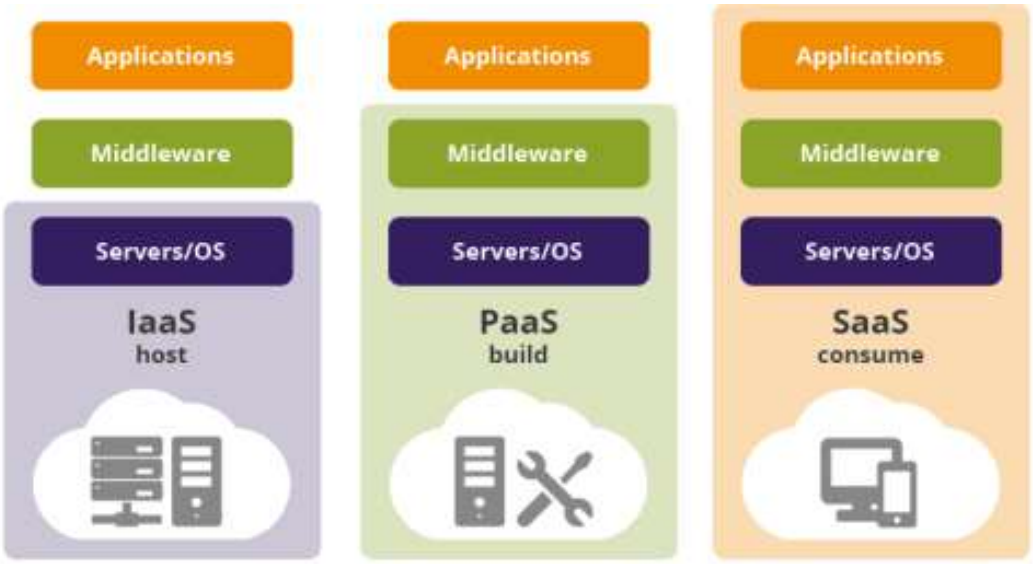
Penelitian Oleh	Parameter Penelitian				
	Objek	Metode	Resource	Database	Parameter
[10]		balancing			time
Apriliansyah [11]	Web server	Load balancing	Lokal	Mysql	Throughput, respone time, request per second, downtime, dan CPU utilization
Alwin Fauzan	Learning Management System (LMS)	Docker container dan load balancing	Cloud	Mariadb	Throughput, delay, pakcet loss, jitter, dan CPU Usage

Pada Tabel 2.1 menjelaskan mengenai rangkuman keterkaitan dengan penelitian sebelumnya. Sejumlah penelitia n telah dilakukan dalam berbagai bidang dengan fokus yang berbeda-beda. Sebagai contoh, penelitian yang dilakukan oleh Suryanto [7] berfokus pada bidang E-learning. Dalam penelitiannya, Suryanto menggunakan metode *docker container* dan sumber daya lokal, serta *database* MySQL. Parameter-parameter yang diukur meliputi CPU *utilization*, memori, *latency*, dan *throughput*. Di sisi lain, Abdillah [8] melakukan penelitian pada *web server* dengan pendekatan *docker container* dan *load balancing*. Penelitian ini melibatkan sumber daya lokal dan *database* MySQL, dengan parameter pengujian mencakup pengambilan data uji, *Request Per Second* (RPS), dan *Time Per Request* (TPR). Penelitian lainnya, seperti yang dilakukan oleh Riskiono et al. [9], Sofyan et al. [10], dan Apriliansyah [11], juga mengevaluasi berbagai aspek kinerja sistem menggunakan pendekatan dan parameter yang berbeda-beda. Selain itu, Alwin Fauzan melakukan penelitian pada *Learning Management System* (LMS) dengan memanfaatkan *docker container* dan *load balancing* pada sumber daya *cloud computing*, dan menggunakan *database* MariaDB. Parameter yang dianalisis mencakup *throughput*, *delay, packet loss*, *jitter* dan CPU *usage*. Dengan demikian, semua penelitian ini secara kolektif memberikan wawasan yang berharga mengenai performa sistem dan aplikasi dalam berbagai konteks.

2.2 Dasar Teori

2.2.1 *Cloud Computing*

Cloud computing adalah sebuah model komputasi dimana sumber daya seperti *processor* / *computing power*, *storage*, *network*, dan *software* dijalankan sebagai layanan melalui media jaringan, bahkan dapat diakses di tempat manapun selama terkoneksi dengan internet. Konsep *cloud computing* adalah sistem yang dapat menambahkan kemampuan infrastrukturnya secara dinamis, tanpa perlu mengeluarkan uang untuk membangun infrastruktur baru, tanpa perlu menambah dan memberikan pelatihan kepada pengelola infrastrukturnya, tanpa perlu membeli lisensi perangkat lunak. Infrastruktur *cloud computing* sangat andal dan dibangun dengan teknologi virtualisasi [12]. Sumber daya seperti penyimpanan, komputasi atau jaringan, disediakan kepada pengguna kedalam layanan berupa infrastruktur, *platform* atau perangkat lunak. Infrastruktur *cloud computing* sangat andal dan dibangun dengan teknologi virtualisasi. Model *cloud computing* yang diterapkan pada internal perusahaan disebut *private cloud* dan *cloud computing* yang disediakan ke publik oleh *provider cloud* disebut *public cloud* [13]. Secara umum *cloud computing* dibagi menjadi tiga model pelayanan dasar, yaitu *Software as a service* (SaaS), *Platform as a Service* (PaaS), dan *Infrastructure as a Service* (IaaS). Ketiga model tersebut dijelaskan pada Gambar 2. 1 sebagai berikut [14].



Gambar 2. 1 Model pelayanan dasar *cloud computing*

1. *Software as a Service* (SaaS) merupakan kelanjutan evolusi konsep ASP (*Application Service Provider*). SaaS memberi kemudahan bagi penggunanya untuk bisa memanfaatkan sumber daya perangkat lunak dengan cara berlangganan tanpa mengeluarkan investasi baik untuk in *house development* maupun pembelian lisensi. SaaS merupakan model aplikasi *cloud* yang memanfaatkan *web-based interface* yang diakses melalui *web browser*. Contohnya *Goggle Docs* yang merupakan aplikasi perangkat *office*, sehingga pengguna dapat mengolah dokumen tanpa harus menginstal *microsoft office*.
2. *Platform as a Service* (PaaS) merupakan layanan penyedia modul-modul siap pakai yang dapat digunakan untuk mengembangkan sebuah aplikasi yang hanya bisa berjalan di atas *platform* tersebut. Layanan pada PaaS menyediakan tempat untuk membuat dan menyebarkan aplikasi tanpa perlu tahu berapa banyak *processor* atau memori yang dibutuhkan untuk aplikasi tersebut. Dalam arti lain PaaS menawarkan layanan yang lebih dari sekadar penyimpanan data. Contoh layanan PaaS adalah *Google AppEngine* yang menawarkan layanan bagi pengguna untuk mengembangkan dan *hosting* aplikasi *web*.
3. *Infrastructure as a Service* (IaaS) berada satu level di bawah PaaS. IaaS merupakan layanan yang "menyewakan" sumber daya teknologi informasi dasar meliputi sistem operasi, *processing power*, media penyimpanan, *memory*, kapasitas jaringan dan layanan-layanan lainnya, yang dapat digunakan oleh penyewa untuk menjalankan aplikasi yang dimilikinya. Contoh yang menawarkan layanan IaaS adalah *Amazon*. Pengguna diberi kebebasan melakukan berbagai kegiatan ke *server* seperti menginstal *software*, konfigurasi ijin akses dan *firewall*.

Terdapat lima elemen mendasar pada *cloud computing* sebagai berikut [15].

1) *On-demand self-service*

   Pengguna *cloud computing* dapat menentukan sumber daya komputasi yang dibutuhkan seperti *database*, jaringan, CPU, *storage*, dll secara otomatis atau melayani diri sendiri.

2) *Broad network access*

   Sumber daya *cloud computing* disebarkan melalui jaringan internet serta dapat diakses oleh pengguna *cloud computing* di mana saja dan menggunakan *platform* apa saja seperti laptop, *handphone*, dll. Sumber daya dapat diakses melalui sebuah situs.

3) *Resource pooling*

   Penyedia layanan mengumpulkan sumber daya komputasi yang dibutuhkan pengguna dengan model *multi-tenant*. Model ini bisa berupa *server* fisik atau virtual. Pada *cloud computing* sumber daya berada pada data *center*, dan sumber daya disebarkan kepada pengguna di tempat yang berbeda-beda.

4) *Rapid elasticity*

   Sumber daya komputasi yang digunakan pengguna dapat ditambahkan apabila kebutuhan meningkat, dan dapat dilepaskan ketika tidak lagi dibutuhkan. Oleh karena itu, penyediaan sumber daya komputasi jadi tidak terbatas dalam menangani permintaan yang terus meningkat.

5) *Measured service*

   Penggunaan sumber daya komputasi perlu adanya *monitoring* untuk memantau penggunaan. Sistem ini dapat melihat berapa sumber daya komputasi yang terpakai, banyaknya layanan yang dipakai dan biaya yang dikeluarkan.

Terdapat empat model penyebaran *cloud computing* yang telah didefinisikan komunitas *cloud computing* sebagai berikut: [15].

1) *Private cloud*

   Pada model penyebaran ini infrastruktur dioperasikan untuk keperluan perusahaan serta dikelola oleh perusahaan itu sendiri atau pihak ketiga. Tujuan memanfaatkan *private cloud* untuk memanfaat sumber daya internal yang ada dan aspek keamanan dan kerahasiaan data perusahaan. Akademisi sering memanfaatkan *private cloud* sebagai penelitian dan pengajaran.

2) *Community cloud*

   Beberapa perusahaan membangun infrastruktur *cloud computing* secara bersama-sama untuk digunakan bersama. *Community cloud* akan meningkatkan skalabilitas ekonomi. Infrastruktur dapat dibangun oleh pihak

ketiga atau perusahaan yang mampu membangun infrastruktur *cloud computing*.

3) *Public cloud*

   Model ini adalah model penyebaran yang dominan digunakan oleh pengguna *cloud computing. Public cloud* digunakan untuk pengguna *cloud computing* dan *cloud provider* memiliki hak penuh atas *cloud computing* serta bertanggung jawab atas data center yang diperlukan sebagai sumber daya komputasi. *Public cloud* memiliki kebijakan terhadap profit, biaya. Beberapa yang populer seperti *Amazon Web Service* (AWS), *Microsoft Azure*, *Google Cloud Platform* (GCP).

4) *Hybrid cloud*

   Model ini merupakan kombinasi dari dua atau lebih model *penyebaran cloud computing private*, *public*, atau *community*. Model ini digunakan perusahaan untuk mengoptimalkan sumber daya perusahaan dan meningkatkan kompetensi mereka.

2.2.2 *Docker*

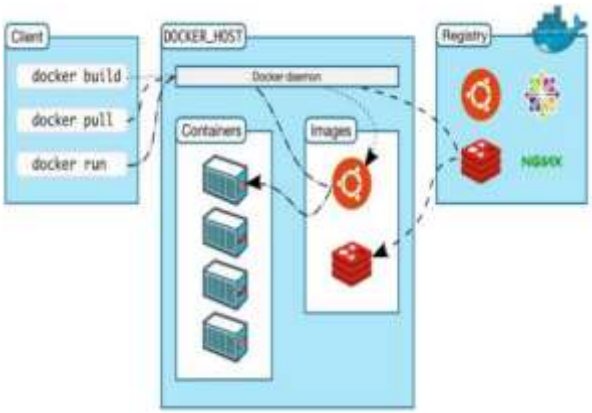
Docker adalah sebuah *peoject open-source* yang menyediakan *platform* terbuka untuk membangun, mengemas, dan menjalankan aplikasi dalam lingkungan terisolasi yang disebut *container*. Keuntungan utama *docker* dalam *deployment* aplikasi *web* adalah fleksibilitas dan efisiensi yang ditawarkannya. Dengan *docker*, aplikasi dapat dikemas bersama dengan semua dependensinya, termasuk sistem operasi, perpustakaan, dan konfigurasi yang dibutuhkan, dalam satu *container*. *Container docker* memiliki sistem yang lebih ringan karena sistem *docker* tidak membawa keseluruhan operasi melainkan berbagi sistem dengan *host* [16]. Dibandingkan dengan mesin virtual, *docker* memiliki keunggulan sebagai berikut [16]:

1. *Docker* memiliki kinerja dan efisiensi yang lebih tinggi daripada metode virtualisasi tradisional. Tidak seperti virtualisasi lapisan perangkat keras dari mesin virtual, *docker* tidak memiliki emulasi perangkat keras, dan mengimplementasikan virtualisasi pada level sistem operasi.
2. *Docker* memiliki lebih sedikit lapisan abstraksi dan tidak memerlukan OS (Sistem Operasi) tambahan dan dukungan *hypervisor*. Berkat ini, *docker*

memiliki pemanfaatan sumber daya yang lebih baik. Biasanya, ada ribuan *container docker* yang berjalan pada satu mesin yang hanya dapat menampung sejumlah kecil mesin virtual. Karena *docker* ringan, waktu *startup* hanya membutuhkan beberapa detik, jauh lebih cepat dibandingkan dengan beberapa menit yang dibutuhkan mesin virtual.

3. *Docker* dapat berjalan di hampir semua *platform*, yang membuat *docker* memiliki mobilitas dan skalabilitas yang lebih baik. Selain itu, mudah untuk proses penyebaran dan pemeliharaan.

Arsitektur docker mempunyai beberapa komponen yaitu terdiri dari *docker daemon*, *docker client*, *docker images*, *docker container* dan *docker registry*. *Docker* menggunakan teknologi *client-server* untuk menghubungkan antara *docker client* dan *docker daemon.* Berikut merupakan Gambar 2. 2 mengenai arsitektur *docker* [17].



Gambar 2. 2 Arsitektur *docker*

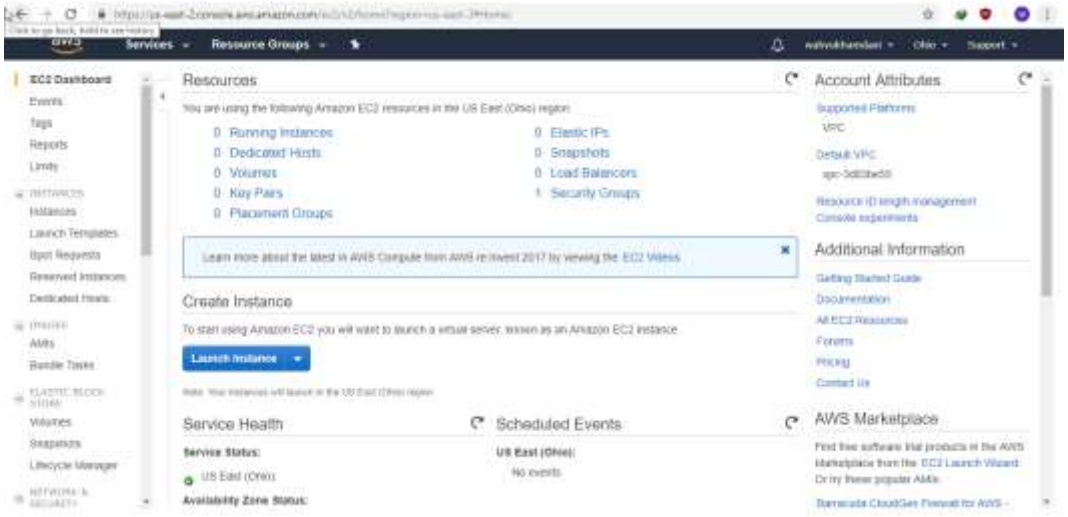
Docker memiliki banyak komponen yang saling bergantungan satu sama lain, sehingga jika ada komponen yang tidak tersedia maka akan mempengaruhi kinerja *docker* [17].

Berikut merupakan komponen yang terdapat pada *docker* [17]:

1. *Docker Daemon* adalah sebuah *service* yang dijalankan di dalam *host* dalam *Operating System* (OS). Berfungsi untuk membangun, mendistribusikan, dan menjalankan *container docker*. Pengguna tidak dapat langsung menggunakan *docker daemon* karena harus menggunakan *docker client* sebagai perantara atau CLI.

2. *Docker Client* adalah kumpulan perintah untuk menjalankan *docker container*. Misalnya membuat *container*, *start/stop container*, menghapus, dan sebagainya yang ada di *docker daemon*.
3. *Docker Image* merupakan *read-only template* untuk menjalankan *container*. *Template* ini berupa OS atau OS yang di-*install* berbagi aplikasi. *Docker image* berfungsi untuk membuat *docker container*, dengan hanya 1 *docker image* dapat membuat banyak *docker container*.
4. *Docker Container* merupakan sebuah *image* yang bersifat *read-write*, *container* berjalan diatas *image*. *Docker container* juga disebut *folder*. Setiap perubahan yang disimpan pada *container* akan menyebabkan terbentuknya *layer* baru di atas base *image*.
5. *Docker Registry* merupakan *repository* disrtibusi kumpulan *docker images* yang bersifat *private* maupun *public* yang dapat diakses melalui *docker hub.*

2.2.3 *Amazon Web Service* (AWS)



Gambar 2. 3 Tampilan *Amazon Web Service* (AWS)

Pada Gambar 2.3 merupakan tampilan *Amazon Web Services* (AWS). AWS adalah *platform* penyedia layanan *cloud computing* yang disediakan oleh *amazon* sejak tahun 2002. *Amazon* sebelumnya lebih terkenal dengan toko buku online-nya. Meski demikian, *Amazon* mengembangkan dirinya menjadi *Amazon Web Services* (AWS) yang menyediakan layanan komputasi awan, dimana disetiap fungsi yang ada di dalamnya dapat diakses menggunakan panggilan *web services*.

Layanan-layanan *Amazon Web Services* dapat dikelompokkan menjadi 5 bagian besar, yaitu [18]:

1. Layanan komputasi

   Layanan ini di khususkan untuk memberi infrastruktur bagi pengguna yang ingin menggunakan *Amazon* untuk melakukan komputasi seperti *server* dan *clustered server*. Layanan komputasi yang disediakan yakni :

   a. *Amazon Elastic Compute Cloud* (EC2) adalah platform komputasi berupa *virtual computer* yang dapat dikustomisasi atau dikembangkan. Untuk penggunaanya sendiri sangat mudah untuk dilakukan.

   b. *Amazon Elastic Map Reduce* adalah layanan yang akan membantu penggunanya melakukan analisis data. Data-data tersebut dapat dikonversikan menjadi sebuah hasil analisis yang dapat digunakan dalam sistem pengambil keputusan.

   c. *Elastic Load Balancing* adalah layanan yang menjadi bagian dari *Amazon* EC2. Layanan ini berfungsi untuk menyeimbangkan beban antara *instance-instance* yang terdapat pada *Amazon* EC2.

2. Layanan Penyimpanan

   Layanan yang memberi infrastruktur untuk pengguna yang ingin menggunakan *Amazon* untuk melakukan penyimpanan. Layanan ini dapat digunakan oleh pengguna sebagai media *backup* maupun *Content Delivery Network* (CDN). Layanan penyimpanan yang disediakan yakni :

   a. *Amazon Simple Storage Service* (S3) adalah layanan media penyimpanan yang sangat aman dan dapat diandalkan dengan harga yang tergolong murah. Layanan ini terintegrasi dengan layananlayanan *Amazon* lainnya, salah satunya *Amazon* EC2. *Amazon* S3 ini dapat di-*install* di komputer, sehingga dapat menjadikan Amazon S3 menjadi *shared folder* maupun *Network Attached Storage*.

   b. *Amazon Elastic Block Store* (EBS) adalah layanan yang menjadi satu paket dengan *Amazon* EC2. EBS *storage* ini menjadi pengganti *hard drive* dari *Amazon* EC2. EBS ini juga media yang disimpan di atas *Amazon* S3.

c. AWS *Storage Gateway* adalah layanan penyimpanan untuk perusahaan berskala besar. Layanan ini membutuhkan *Vmware HyperX* dengan *requirements* yang cukup tinggi. Layanan ini sangat cocok untuk perusahaan yang memiliki banyak data sensitif dan butuh *offsite backup*.

d. *Amazon CloudFront* adalah layanan untuk distribusi konten ke berbagai lokasi *server Amazon*. Layanan ini banyak digunakan untuk aplikasi *web* sehingga kecepatan *load* untuk konten menjadi jauh berkurang, sebab konten diambil dari lokasi terdekat dari pengguna yang sedang melakukan akses ke *website*.

3. Layanan Basis Data

   Layanan ini dikhususkan untuk basis data, dimana basis data tersebut disimpan di-*cloud*, dan dapat diakses darimana saja secara aman, cepat dan terpercaya. Layanan basis data yang disediakan yakni :

   a. *Amazon Relational Database Service* (RDS) adalah layanan *server* basis data dimana data dan *server* akan berada di *cloud* yang akan menjadi kualitas koneksi, kecepatan, keamanan, dan kehandalan.

   b. *Amazon DynamoDB* adalah layanan *server* basis data yang NoSQL dengan kualitas terbaik di segala aspek dan yang pastinya mudah dalam melakukan *setup* dan konfigurasi. Layanan ini memiliki kemudahan skalabilitas sehingga data dapat berkembang dan menyusut sesuai dengan keperluan.

   c. *Amazon SimpleDB* adalah layanan *server* basis data yang NoSQL yang mirip dengan *Amazon DynamoDB* dalam segi kualitas namun dengan skala yang lebih kecil.

   d. *Amazon ElastiCache* adalah layanan *memory cache* di atas *cloud*. Layanan ini dapat meningkatkan performa dari aplikasi *web* dengan menyimpan *cache* dan tidak membebani *server web* aplikasi itu sendiri.

4. Layanan Jaringan

   Layanan ini di khususkan untuk mengatur jaringan antara layanan-layanan yang di dalam *cloud* maupun diluar *cloud*.

   a. *Amazon Route* 53 adalah layanan untuk *Domain Name Server* (DNS), layanan ini memberikan akses yang cepat dan aman untuk *domain* dari

aplikasi *web* yang telah dimiliki. Layanan ini memiliki fitur *load balance* jika memiliki lebih dari satu *server*.

b. *Amazon Virtual Private Cloud* adalah layanan untuk memudahkan pembuatan *private cloud* dengan menggabungkan layanan yang ada dalam AWS.

5. Layanan Aplikasi

   Layanan aplikasi ini disediakan oleh *Amazon* untuk melengkapi layanan-layanan yang lainnya. Layanan-layanan yang disediakan seperti :

   a. *Amazon Cloud Search* – Aplikasi Pencarian
   b. *Amazon Simple Workflow Service* (SWF) – Aplikasi *Workflow*
   c. *Amazon Simple Queue Service* (SQS) – Aplikasi Antrian
   d. *Amazon Simple Notification Service* (SNS) – Aplikasi Notifikasi
   e. *Amazon Simple Email Service* (SES) – Aplikasi *Email Server*.

2.2.4 *Learning Management System* (LMS)

Learning Management System (LMS) adalah sebuah teknologi yang dirancang dan dikembangkan untuk mengelola maupun mendukung pembelajaran perkuliahan, dimana pada LMS dapat difungsikan untuk mendistribusikan materi hingga proses kolaborasi antara dosen dan mahasiswa. LMS akan memungkinkan untuk mengelola setiap aspek kursus, mulai dari pendaftaran, administrasi, dokumentasi, pelaporan, otomatisasi dan penyampaian kursus pendidikan, program pelatihan, atau program pembelajaran dan pengembangan hingga penyimpanan hasil tes, dan juga memungkinkan untuk menerima tugas secara digital. Contoh aplikasi yang menunjang LMS adalah *google classroom, moodle, schoology, edmodo,* dan lain-lain [19].

2.2.5 *Moodle*

Moodle merupakan singkatan dari *Modular Object Oriented Dynamic Learning Environment*. *Moodle* adalah sebuah teknologi yang digunakan untuk menunjang sistem manajemen pembelajaran secara *online*. *Moodle* merupakan *platform* pembelajaran yang bisa diakses secara terbuka dan digunakan oleh banyak institusi pendidikan diseluruh dunia. Kehadirannya yang luas ini memberikan keyakinan bahwa penelitian yang dilakukan menggunakan *moodle* memiliki validitas dan dapat dibandingkan dengan hasil-hasil sebelumnya dari

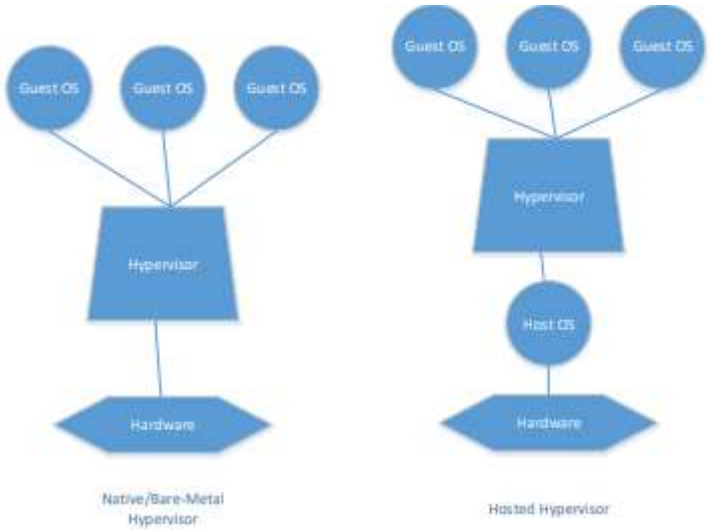
berbagai institusi pendidikan. *Platform* ini memungkinkan pengguna untuk masuk kedalam "ruang kelas" *virtual* untuk mengakses materi-materi pembelajaran. *Moodle* sangat mendukung pembelajaran elektronik dalam berbagai macam format materi pembelajaran yaitu dalam bentuk teks, portofolio, animasi, audio dan video dan lainnya. Dengan menggunakan format ini, pengajar dapat menyampaikan materi pembelajaran melalui *e-learning* dari jarak jauh. Konsep pembelajaran ini menggunakan sistem belajar mengajar yang tidak terbatas ruang dan waktu. Seorang pengajar dapat memberikan materi dari mana saja. Begitu juga bagi pelajar dapat mengakses materi dimana dan kapan saja [20].

Adapun kelebihan yang dimiliki oleh *moodle*, yaitu [20] :

1. Cocok untuk pembelajaran *online*.
2. Sederhana, ringan dan efisien.
3. Mudah di instal pada banyak program yang bisa mendukung PHP dan membutuhkan satu *database* .
4. Mendukung 1000 pelajaran lebih.
5. Mempunyai keamanan yang baik.
6. Paket bahasa tersedia penuh untuk berbagai bahasa, lebih dari 45 bahasa.

2.2.6 *Hypervisor*

Hypervisor adalah sebuah *firmware* yang bertugas membuat, mengatur, menjalankan, dan memonitor sebuah mesin virtual. *Hypervisor* diklasifikasikan menjadi 2 jenis tergantung tempat di mana *hypervisor* berdiri [21].



Gambar 2. 4 Jenis-jenis *hypervisor*

Pada Gambar 2. 4 menjelaskan mengenai 2 jenis *hypervisor*, yaitu [21]:

1. *Hypervisor type* 1 (*native* atau *bare-metal*), adalah sebuah perangkat lunak khusus yang beroperasi secara langsung di atas perangkat keras fisik tanpa memerlukan sistem operasi *host* tambahan. Ini berarti *hypervisor tipe* 1 bekerja secara efisien sebagai lapisan virtualisasi langsung di atas *hardware*. Contoh *hypervisor tipe* 1 meliputi VMware ESX/ESXi dan Microsoft Hyper-V 2008/2012. Kedua produk ini digunakan secara luas dalam lingkungan virtualisasi bisnis dan data *center*, karena kinerja dan keamanan yang tinggi serta kemampuan untuk mengelola lingkungan virtual dengan efisien.
2. *Hypervisor type* 2 (*hosted hypervisor*), jenis ini hanya dapat mengakses perangkat keras pada suatu mesin fisik melalui sebuah sistem operasi di bawahnya. Contoh dari *hypervisor* jenis ini adalah VMware Workstation, VMware Player, VMware Fusion, Microsoft Virtual PC, Oracle VirtualBox dan lain sebagainya. Perbedaan tempat sebuah *hypervisor* itu berjalan merupakan faktor yang cukup menentukan performa dari sebuah mesin virtual yang dijalankan. Performa dari sebuah mesin virtual yang terdapat dalam baremetal terbukti lebih baik dari pada *hosted hypervisor*. Hal ini dikarenakan mesin virtual tersebut memiliki akses langsung terhadap perangkat keras dari sebuah mesin fisik. Mesin virtual tersebut dapat menggunakan sumber daya lebih bebas karena tidak terpotong oleh sebuah sistem operasi yang berjalan di bawahnya. Sedangkan pada *hypervisor* tipe 2 dapat dengan cepat membuat dan menghapus mesin virtual (VM) tanpa memerlukan perubahan besar pada perangkat keras atau konfigurasi *host*. Ini memberikan fleksibilitas dan mobilitas yang lebih besar untuk pengujian dan pengembangan dibandingkan dengan *hypervisor tipe* 1.

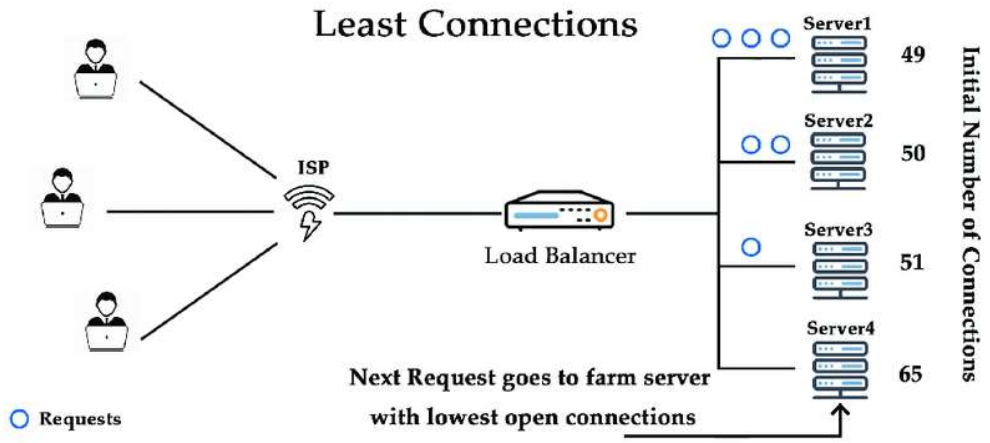
2.2.7 *Load Balancing*

Load balancing adalah proses pendistribusian beban terhadap beberapa *server* atau untuk meminimalkan waktu respon dari *server*, dan menghindari kelebihan beban pada *server*. *Load balancing* juga dapat melakukan berbagai macam fungsi seperti *traffic engineering*, penyeimbang beban, serta fungsi pemindahan jalur trafik. Dalam penerapannya, *load balancing* memiliki berbagai

macam algoritma penjadwalan yang dapat digunakan untuk mendistribusikan beban kepada setiap *server* yang berada di dalam sebuah *server* [22].

2.2.8 Algoritma *Least Connection*

Algoritma *least connection* adalah algoritma yang melakukan beban berdasarkan banyaknya koneksi yang sedang dilayani oleh sebuah *server*. *Server* dengan koneksi paling sedikit akan diberikan beban, berikutnya *server* dengan koneksi paling banyak akan dialihkan bebannya ke *server* lain yang bebannya lebih rendah. Berikut merupakan Gambar 2.5 mengenai konsep algoritma *least connection* [23].



Gambar 2. 5 Konsep algoritma *least connection*

Algoritma *least connection* melihat status beban *real-time* dari *backend node server* dengan melihat nomor koneksi yang tercatat pada *load balancer*, lalu membagikan beban berdasarkan koneksi ke *server* yang paling sedikit. Diberikan asumsi bahwa sebuah sistem memiliki *node server* S0, S1, S2, … , Sn−1, C(Si) yang berarti jumlah koneksi pada *server* i dan S_m yang berarti *server* yang ditentukan untuk melayani permintaan yang baru masuk. Algoritma *least connection* dapat dijabarkan sebagai berikut [24].

Input: menetapkan pilihan terakhir yang akan ditetapkan pada *node server* dan total data *server*.

Output: memilih *server* yang akan dibagikan beban.

Least_Connection_Algorithm(node m, Servers n)

1) Untuk melintasi cluster server

2) Menentukan apakah server m telah berfungsi dengan benar, jika tidak berfungsi dengan benar maka kembali ke langkah satu dan beban kerja dipindah ke server berikutnya. Jika berfungsi, lanjutkan ke langkah tiga

3) For (melintasi cluster server dari server m +1);

4) Temukan server yang mendapatkan koneksi paling sedikit;

5) Temukan server yang tepat, dikembalikan ke S_m;

6) Kembali ke NULL jika tidak mendapatkan server yang tepat.

Code Description:

```
Least_Connection_Algorithm(node m, Servers n)
{
  for(m=0;m<n;m++)
  {
    if (Sm is alive)
    {
      for(i=m+1;i<n;i++)
        {
          if(C(Si) < C(Sm))
          m=i;
        }
      return Sm
    }
  }
return NULL;
}
```

2.2.9 *Mariadb*

Mariadb merupakan *Relational Database Management System* (RDBMS) yang digunakan untuk menyimpan dan memanajemen data. RDBMS ini merupakan perangkat lunak (*software*) yang dipakai untuk membangun basis data yang berbasis komputerisasi. Secara umum baik perintah, fungsi, maupun

tampilan mempunyai kemiripan antara MySQL dengan *Mariadb*. Beberapa bahasa pemrograman yang bisa dapat digunakam pada *mariadb* antara lain C++, *Perl*, *Java*, PHP, dan *Python*. Program yang telah dibuat tersebut yang nantinya akan digunakan untuk proses input data ke dalam database [25].

2.2.10 *Fortigate Firewall*

Fortigate merupakan suatu sistem keamanan yang diluncurkan oleh perusahaan *Fortinet*. *Fortigate* sebagai perangkat yang menjamin keamanan jaringan secara keseluruhan dan berfungsi sebagai *gateway* dan *router* bagi jaringan LAN sehingga tidak dibutuhkan *router* atau perangkat tambahan *load balancing* lain bila ada lebih dari satu koneksi WAN. *Firewall* pada dasarnya dimaksudkan untuk melindungi jaringan internal terhadap berbagai gangguan ataupun serangan yang berasal dari luar, *firewall* melindungi perangkat *router* dan *client-client* yang terhubung pada suatu jaringan. *Firewall* dapat menyeleksi paket yang melewatinya, berdasarkan aturan yang dibuat administrator. *Next-Generation Firewall* atau sering disebut GFW lebih kuat dibandingkan dengan *Traditional Firewall*. NGFW memiliki kemampuan *Traditional Firewall* dan juga memiliki beberapa fitur tambahan untuk menangani lebih banyak variasi ancaman. *Firewall* ini disebut "*Next-Generation*" untuk membedakan dari firewall lama atau biasa disebut *Traditional Firewall* yang tidak mempunyai kemampuan tambahan tersebut. *Next-Generation Firewall* adalah bagian dari teknologi *firewall* generasi ketiga, yang menggabungkan *firewall* tradisional dengan fungsi penyaringan perangkat jaringan lainnya, seperti *Application Firewall* yang menggunakan *Deep Packet Inspection* (DPI) *in-line, Intrusion Prevention System* (IPS). *Next-Generation Firewall* adalah solusi *firewall* dan keamanan jaringan yang juga memiliki fitur-fitur untuk *load balancing*. Penggunaan *fortigate* dalam penggunaan *load balancing* memiliki beberapa alasan: [26]

1. Pemantauan dan Analisis

   Fortigate menyediakan kemampuan pemantauan lalu lintas yang mendalam, yang memungkinkan untuk melihat statistik dan data lalu lintas secara *real-time*.

2. Distribusi Beban

Fortigate dapat melakukan distribusi lalu lintas secara cerdas berdasarkan beberapa faktor, seperti beban *server*, kesehatan *server*, jenis lalu lintas, dan lainnya.

3. Integrasi dengan Keamanan

   Pada saat menggunakan fitur keamanan *fortigate*, integrasi antara *traffic load balancing* dan keamanan dapat memberikan lapisan perlindungan tambahan. Misalnya, *fortigate* dapat mengarahkan lalu lintas yang terindikasi mencurigakan atau berbahaya ke sistem deteksi ancaman.

2.2.11 *Apache Benchmark*

Apache benchmark adalah sebuah *tool* dari *Apache organization* yang digunakan untuk mengukur performansi pada *Hypertext Transfer Protocol* (HTTP) *web server*. *Tool* ini digunakan untuk menghitung berapa banyak *request per second* yang dapat di layani oleh *web server* yang digunakan. Beberapa fitur dari *apache benchmark* seperti *open source*, *simple command line, platform independent, load* and *performance test, not extensible*. *Apache benchmark* dapat digunakan untuk menguji performa dari *web server* dengan berbagai batasan pengujian seperti *transfer rate* dan *request per second* [27].

2.2.12 *Quality Of Service* (QOS)

Quality of Service / kualitas layanan adalah kemampuan suatu jaringan untuk menyediakan layanan yang baik dengan menyediakan *bandwidth*, mengatasi *jitter* dan *delay*. Parameter yang tercakup secara umum dalam QoS antara lain *throughput, delay, jitter,* dan *packet loss* [28].

Parameter QoS terdiri dari :

1) *Throughput*

   Throughput merupakan kecepatan transfer data efektif yang diukur dalam *bit* per *second* (bps), dapat dihitung yang ditunjukkan pada persamaan 2.1 [28].

   Rumus untuk menghitung *throughput* adalah :

$$Throughput = \frac{Jumlah\ paket\ data\ yang\ dikirim\ (byte)}{Lama\ pengamatan\ (s)} \quad (2.1)$$

   Diubah ke satuan bit:

   Throughput (bps) = $Throughput\ (byte)\ x\ 8$ (bit)

Pengelompokkan standarisasi *throughput* berdasarkan standar TIPHON ditunjukkan pada tabel 2.2.

Tabel 2. 2 Standarisasi *Throughput*

Kategori	Nilai
Sangat bagus	>2,1 Mbps
Bagus	1200 Kbps - 2,1 Mbps
Sedang	700 Kbps - 1200 Kbps
Buruk	<700 Kbps

2) *Delay* merupakan waktu yang diperlukan beberapa data untuk mencapai tujuannya diseluruh jaringan atau bisa diartikan dengan waktu yang dibutuhkan informasi untuk mencapai tujuannya dan kembali lagi. Dapat dihitung yang ditunjukkan pada persamaan 2.2 [28].
   Rumus untuk menghitung rata-rata *delay* adalah :

$$\text{Rata-rata } delay = \frac{Total\ delay}{Total\ paket\ yang\ diterima} \quad (2.2)$$

Pengelompokkan standarisasi *delay* berdasarkan standar TIPHON ditunjukkan pada tabel 2.3.

Tabel 2. 3 Standarisasi *Delay*

Kategori	Nilai (ms)
Sangat bagus	<150
Bagus	150 – 300
Sedang	300 – 450
Buruk	>450

3) *Packet Loss*
   Parameter ini merupakan suatu kondisi yang menampilkan total paket yang hilang selama pengiriman, hal ini terjadi karena adanya *collision* dan *congestion* pada jaringan [28].

Rumus menghitung *packet loss* ditunjukan pada persamaan 2.3.

$$Packet\ Loss = \frac{(Paket\ data\ dikirim - Paket\ data\ diterima)\ x\ 100\%}{Paket\ data\ yang\ dikirim} \quad (2.3)$$

Pengelompokkan standarisasi *packet loss* berdasarkan standar TIPHON ditunjukkan pada tabel 2.4

Tabel 2. 4 Standarisasi *Packet Loss*

Kategori	Nilai (%)
Sangat bagus	0
Bagus	3
Sedang	15
Buruk	25

4) Jitter

Parameter ini adalah variasi kedatangan paket, *jitter* dapat terjadi dikarenakan adanya variasi-variasi dalam panjang antrian, dalam waktu pengolahan data, dan juga dalam waktu penghimpunan ulang paket-paket diakhir perjalanan *jitter* [28].

Rumus menghitung *jitter* ditunjukan pada persamaan 2.4.

$$Jitter = \frac{Total\ variasi\ delay}{Total\ paket\ yang\ diterima} \quad (2.4)$$

Pengelompokkan standarisasi *jitter* berdasarkan standar TIPHON ditunjukkan pada tabel 2.5.

Tabel 2. 5 Standarisasi *Jitter*

Kategori	Nilai (ms)
Sangat bagus	0
Bagus	0 – 75
Sedang	75 – 125
Buruk	125 – 225

5. CPU *Usage*

   CPU *usage* adalah jumlah sumber daya yang diperlukan dalam melaksanakan proses komputasi pada *server* [28].