BAB 2
KAJIAN PUSTAKA DAN DASAR TEORI

2.1 KAJIAN PUSTAKA

Penelitian yang dilakukan oleh Haris Sandra pada Oktober 2022, meneliti dan membandingkan performa dari dua web server populer, yaitu Openlitespeed dan Apache, dalam menangani *client request*, yang berjudul "*Performance Analysis of Openlitespeed and Apache Web Servers in Serving Client Requests*". Web server bertugas untuk memproses permintaan dari klien dan mengirimkan respons yang sesuai. Penulis melakukan analisis performa dengan menguji kedua web server dengan beban (*load*) yang berbeda dan mengukur waktu yang dibutuhkan untuk memproses permintaan dan mengirimkan respon. Hasil dari analisis menunjukkan bahwa Openlitespeed memiliki performa yang lebih baik dibandingkan dengan Apache dalam menangani *client request*. Peneliti juga menemukan bahwa Openlitespeed lebih unggul dalam hal skalabilitas dan efisiensi sumber daya, sehingga lebih cocok digunakan dalam lingkungan yang membutuhkan tingkat performa tinggi. Sementara itu, Apache memiliki lebih banyak fitur dan kemampuan dibandingkan dengan Openlitespeed, sehingga lebih baik digunakan dalam lingkungan yang membutuhkan fleksibilitas dan kemampuan tambahan. Secara keseluruhan, jurnal ini menyimpulkan bahwa kedua web server memiliki kelebihan dan kekurangan masing-masing dan performa mereka bergantung pada lingkungan dan kebutuhan pemakaian. Oleh karena itu, pemilihan web server yang tepat harus didasarkan pada evaluasi yang tepat terhadap lingkungan dan kebutuhan pemakaian[4].

Penlitian lain juga di lakukan oleh Albert Yakobus Chandra yang berjudul "Analisis Performansi Antara Apache & Nginx Web Server Dalam Menangani *Client Request*" pada November 2019, melakukan penelitian terhadap performa dua web server yaitu Apache dan Nginx, dalam menangani *client request*. Web server bertugas untuk memproses permintaan dari klien dan mengirimkan respons yang sesuai. Hasil dari analisis menunjukkan bahwa Nginx memiliki performa yang lebih baik dibandingkan dengan Apache dalam menangani *client request*. Penulis juga mengungkapkan Nginx memiliki kelebihan dalam hal skalabilitas dan efisiensi

sumber daya, sehingga lebih baik digunakan dalam lingkungan yang membutuhkan tingkat performa tinggi. Peneliti juga mencatat bahwa Apache memiliki lebih banyak fitur dan kemampuan dibandingkan dengan Nginx, sehingga lebih baik digunakan dalam lingkungan yang membutuhkan fleksibilitas dan kemampuan tambahan. Secara keseluruhan, jurnal ini memberikan gambaran tentang performa kedua web server yang populer dan menunjukkan bahwa pemilihan web server yang tepat harus didasarkan pada analisis yang tepat dan evaluasi terhadap kebutuhan dan lingkungan pemakaian. Oleh karena itu, penentuan web server yang tepat harus dilakukan dengan hati-hati untuk memastikan kinerja yang optimal dan stabil dalam lingkungan pemakaian[3].

Pengembangan *tool deployment* semakin populer, dari yang *deploy* secara manual hingga dengan *automation* baik secara *baremetal*/fisik ataupun secara *container*. Kubernetes merupakan salah satu *tool* yang dapat mengelola *container* atau *container orchestration*. Penelitian yang dilakukan oleh Imron Rosyadi, Shoffin Nahwa Utama, dan Oddy Virgantara Putra pada tahun 2019 berjudul "*Implementation Autoscalling Container Web Server Using Kubernetes Proxmox-based on Server University of Darussalam Gontor*". Tujuan dari penelitian ini adalah untuk mengimplementasikan *autoscaling* pada web server menggunakan Kubernetes Proxmox pada server Universitas Darussalam Gontor. Dalam penelitian ini, para peneliti menggunakan teknologi container untuk menjalankan web server wordpress pada server, dengan tujuan untuk mengatasi *client request* yang tinggi terhadap web server agar meningkatkan efisiensi penggunaan sumber daya. Hasil penelitian menunjukkan bahwa implementasi *autoscaling* pada web server menggunakan Kubernetes pada Proxmox server Universitas Darussalam Gontor berhasil meningkatkan performa web server dalam menangani *client request* secara otomatis, dibuktikan dengan penurunan penggunaan CPU 13,01%, *throughtput* yang lebih besar 4494,34 KB/s dan *response time* yang lebih baik selisih 14,46 *requests*/detik antara sebelum dan sesudah dikonfigurasi *autoscalling*[1].

Teknologi *container* yang di orkrestasi oleh kubernetes, juga dipakai dalam memigrasi suatu layanan web server kedalam bentuk *container*. Berdasarkan penelitian yang dilakukan oleh Muhammad Imran, Valentin Kuznetsov, Katarzyna Maria Dziedziniewicz-Wojcik, Andreas Pfeiffer, Panos Paparrigopoulos, Spyridon

Trigazis, Tommaso Tedeschi dan Diego Ciangottini yang berjudul "*Migration of CMSWEB cluster at CERN to Kubernetes: a comprehensive study*" pada Juni 2021. Penelitian ini, mengimplementasikan kubernetes sebagai orkrestasi container CMSWEB disertai fitur *horizontal pod autoscaling,* yang berdasarkan nilai metrik khusus yang telah evaluasi sebelumya. Hasil penelitian menunjukkan kluster CMSWEB yang ada pada kubernetes dapat meningkatkan efisiensi dalam hal memori, disk dan penggunaan CPU. Studi ini juga menemukan bahwa performa layanan di dalam kluster Kubernetes sangat dipengaruhi oleh beberapa faktor seperti pengaturan memori dan CPU, jumlah *instance* layanan yang berjalan pada pod, dan penerapan *autoscaling*[2]. Berikut tabel 2.1 mengenai rangkuman penelitian yang telah dilakukan sebelumnya.

Tabel 2.1 Rangkuman Keterkaitan dengan Penelitian Sebelumnya

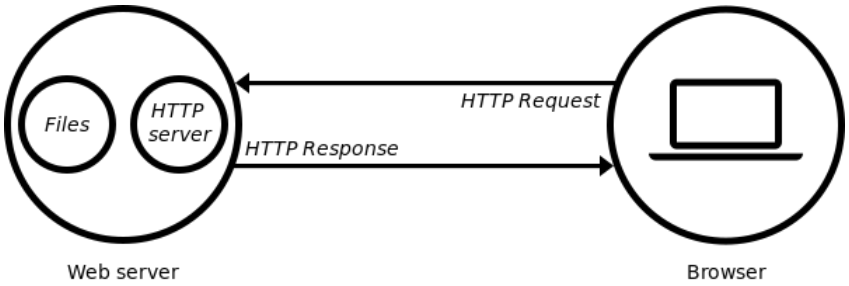
Penelitian Oleh	Parameter Penelitian			
	Objek	Metode	*Resource*	Parameter
Haris Sandra[4]	Openlitespeed dan Nginx	Pagespeed insight, gtmetrix dan webpagetest	Web Server	*Load Time,* dan *Speed Index*
Albert Yakobus Chandra[3]	Apache dan Nginx	Apache Benchmark	Web Server	*Response Time*
Imron Rosyadi dkk. [1]	CMS Wordpress	Horizontal pod *Autoscalling*	Proxmox Server dan Kubernetes	*Response Time*, *Throughput*, CPU *Usage*
Muhammad Imran dkk. [2]	CMSWEB	Horizontal pod *Autoscalling*	Kluster Kubernetes	*Throughput, Latency,* dan *CPU Usage*

2.2 DASAR TEORI

2.2.1 Web Server

Web server merupakan sebuah perangkat lunak yang menyimpan dan mengirimkan konten untuk sebuah situs web seperti teks, gambar, video, dan data aplikasi pengguna yang memintanya. Pengguna yang paling umum adalah program

peramban web atau *web browser* yang meminta data dari situs web ketika pengguna mengklik tautan atau mengunduh dokumen pada halaman yang ditampilkan di peramban**.** Web server berkomunikasi dengan web browser menggunakan HTTP seperti pada gambar 2.1. Konten dari sebagian besar halaman web dikodekan dalam bentuk HTML. Konten bisa bersifat statis seperti teks, gambar atau dinamis seperti nilai mata uang, harga produk. Untuk memberikan konten dinamis, sebagian besar web server mendukung bahasa *scripting server-side* [5].



Gambar 2.1 Proses Komunikasi Web server[6]

Web server memberikan dukungan untuk HTTP dalam berkomunikasi. HTTP secara khusus mengatur bagaimana mentransfer dokumen web terkait (*hypertext*) antara dua *host* (klien) atau lebih. Sebuah protokol adalah seperangkat aturan untuk berkomunikasi antara dua komputer. HTTP merupakan protokol yang bersifat teks dan tidak menyimpan informasi dari komunikasi sebelumnya. Semua perintah dalam HTTP berbentuk teks dan mudah dibaca oleh manusia. Server dan klien HTTP tidak menyimpan ingatan komunikasi sebelumnya. Sebagai contoh, server HTTP tidak dapat mengingat password yang telah dimasukkan atau memori proses transaksi yang tidak lengkap. Oleh karena itu, diperlukan server aplikasi untuk tugas semacam itu. Ketika sebuah file diminta melalui HTTP, klien harus memberikan URL file tersebut. Web server harus membalas setiap permintaan HTTP, setidaknya dengan pesan kesalahan. Pada sebuah web server, server HTTP bertanggung jawab dalam memproses dan menjawab permintaan masuk. Setelah menerima permintaan, server HTTP akan memeriksa apakah URL yang diminta sesuai dengan file yang ada. Apabila sesuai, web server akan mengirimkan isi file kembali ke browser. Namun, jika tidak ada file yang sesuai dengan URL, server akan memeriksa apakah harus menghasilkan file secara dinamis untuk permintaan

tersebut. Jika keduanya tidak mungkin dilakukan, web server akan memberikan pesan kesalahan kepada browser, umumnya adalah 404 *Not Found*[6].

HTTPS merupakan protokol yang lebih aman dari HTTPS. Protokol HTTPS menambahkan lapisan keamanan pada komunikasi antara server web dan *browser* pengguna dengan menggunakan protokol SSL/TLS untuk mengenkripsi data yang ditransmisikan. HTTPS biasanya digunakan untuk melindungi data sensitif seperti login, informasi kartu kredit, dan informasi pribadi lainnya. HTTPS juga membantu melindungi data pribadi pengguna dan mencegah penyebaran *malware*. Namun, penggunaan HTTPS tidak sepenuhnya menjadi standar di seluruh web. Untuk mengimplemntasikan HTTPS, sebuah website harus memiliki sertifikat SSL/TLS dari otoritas sertifikat tepercaya. Sertifikat SSL/TLS memastikan bahwa informasi yang ditransmisikan antara browser dan server web dienkripsi dan tidak dapat disadap oleh pihak ketiga[7].

Web server semakin berkembang, dengan seiringnya banyak para *developer development* perangkat lunak web server. Ada beberapa perangkat lunak untuk mengimplementasikan web server yang dapat diketahui, antara lain:

1. **Nginx**

   Nginx adalah sebuah *software* web server yang memiliki performa tinggi dan bersifat *open-source*. Nginx dirancang untuk mengakomodasi kebutuhan pengguna yang membutuhkan web server dengan kemampuan *load balancing, reverse proxy,* dan *caching*. Nginx didesain untuk dapat diinstal pada berbagai sistem operasi, termasuk Linux, Windows, dan Mac OS X. Nginx juga mendukung protokol-protokol seperti HTTP, HTTPS, SMTP, POP3, dan IMAP. Web server ini menggunakan model *asynchronous event-driven* untuk memproses permintaan, yang membuatnya lebih efisien. Kelebihan utama dari Nginx adalah kemampuan untuk menangani banyak permintaan secara bersamaan dengan kecepatan tinggi. Nginx telah berkembang dan mendukung berbagai komponen, seperti HTTP/2, gRPC, dan *streaming* berbagai format video (HDS, HLS, RTMP, dan lainnya)[8]. gRPC merupakan *framework* komunikasi *remote opensource* yang

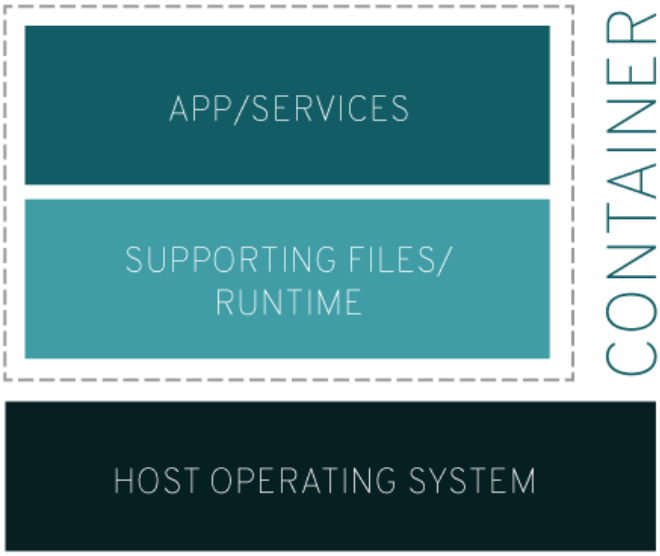
dikembangkan oleh Google. Ini memungkinkan *client* dan server untuk berkomunikasi dengan bahasa yang berbeda [9].

2. **Openlitespeed**

   Openlitespeed adalah sebuah web server yang bersifat *opensource* yang dikembangkan oleh LiteSpeed Web Server Enterprise. Openlitespeed saat ini hanya dapat di instal pada Openlitespeed dirancang dengan menggunakan arsitektur *event-driven* layaknya Nginx yang memungkinkan server untuk menangani lebih banyak koneksi dengan sumber daya yang lebih sedikit. Hal ini memungkinkan Openlitespeed dapat melayani lebih banyak permintaan dalam waktu yang lebih singkat dan menggunakan lebih sedikit sumber daya untuk melakukannya. Openlitespeed juga dilengkapi dengan fitur *caching* yang dapat meningkatkan kecepatan pengiriman konten web dan mengurangi waktu yang dibutuhkan untuk memuat halaman web. Selain itu, Openlitespeed mendukung protokol HTTP/2, yang memungkinkan transfer data yang lebih cepat dan efisien antara server dan *browser*[10].

2.2.2 Container

Container adalah sebuah teknologi virtualisasi yang memungkinkan sebuah aplikasi dijalankan secara terisolasi di atas sistem operasi tanpa harus menggunakan virtual *machine*. Setiap container memiliki lingkungan yang terisolasi sendiri, sehingga masing-masing container dapat memiliki aplikasi atau versi sistem operasi yang berbeda, sesuai pada gambar 2.2 dimana *container* terpisah dari sistem operasi. Teknologi *container* menjadi populer seiring dengan perkembangan metode pengembangan aplikasi modern yang membutuhkan *deployment* yang cepat dan konsisten. Pada dasarnya, *container* menggunakan sistem operasi host sebagai basisnya dan berbagi kernel sistem operasi tersebut. Hal ini berbeda dengan teknologi virtualisasi lainnya seperti *hypervisor* yang mengizinkan *multiple* OS berjalan di atas *physical hardware*. Dengan demikian, container dapat dijalankan dengan lebih efisien dan ringan karena tidak perlu menambah beban kernel virtual.



Gambar 2.2 Arsitektur *Container*[11]

Container yang memuat aplikasi berisi dependensi, *libraries* dan paket yang dibutuhkan, sehingga dengan mudah memindahkannya ke *production* tanpa mengganggu sistem operasi yang sudah berjalan. Sebenarnya, isi dari sebuah *image container* adalah paket RPM atau DEB, file konfigurasi, dll. Namun, distribusi *image container* jauh lebih mudah dibandingkan menginstal salinan baru dari sistem operasi[11].

Container semakin populer sebagai alternatif *virtualization*. Pada *container*, *resource* sistem operasi yang sama digunakan untuk menjalankan beberapa aplikasi secara terisolasi pada lingkungan yang sama, tanpa memerlukan hypervisor seperti pada *virtualization*. Perbedaan utama antara *virtualization* dan *container* adalah cara sumber daya digunakan dan diisolasi. Dalam *virtualization*, *hypervisor* digunakan untuk memisahkan dan mengalokasikan sumber daya seperti CPU, memori, dan penyimpanan. Sedangkan pada *container*, fitur sistem operasi digunakan untuk memisahkan dan mengalokasikan sumber daya. Sehingga, *container* lebih ringan dan lebih cepat dibandingkan dengan virtualisasi. Adapun salah satu *container runtime* sebagai berikut

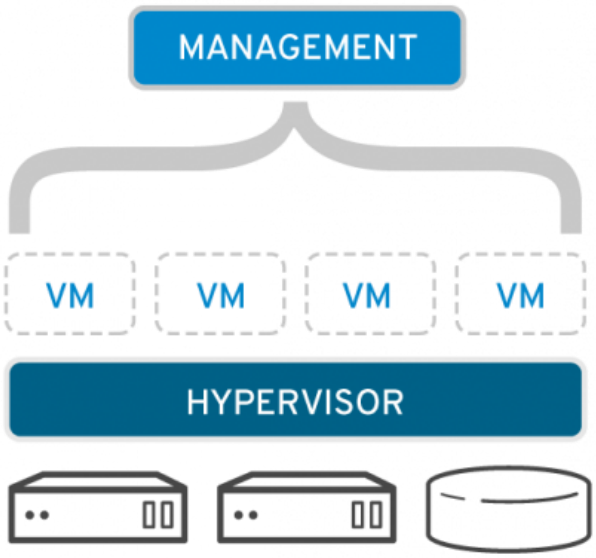
1. **Containerd**

   Containerd merupakan sebuah *runtime container* yang fungsinya adalah menyediakan lingkungan yang portabel, dan mudah ditingkatkan (*scale*) untuk menjalankan *container* di sistem operasi Linux. *Runtime* pada

containerd bertanggung jawab untuk membuat, menjalankan, menghentikan, dan mengelola container. Containerd menyediakan API yang berguna dalam mengendalikan operasi *container* seperti membuat, menghapus, dan menjalankan container. Containerd juga berperan dalam untuk mengelola *snapshot* dari sistem file yang ada pada container. Ini memungkinkan container untuk berbagi atau menggunakan *snapshot* file yang sama. Containerd menyediakan layanan pengelolaan gambar (*image service*) yang memungkinkan pengguna untuk mengunduh, mengunggah, dan mengelola *image container*. Containerd dapat menjalankan proses dalam *container*, termasuk membuat proses, mengalihkan aliran *input/output* (I/O), mengelola isolasi sumber daya, dan mengontrol proses di dalam container. *Sandbox* dalam containerd membuat lingkungan yang aman yang memisahkan proses container dari host dan container lainnya. Ini membantu mencegah proses dalam container untuk berinteraksi secara langsung dengan host atau container lainnya[12].

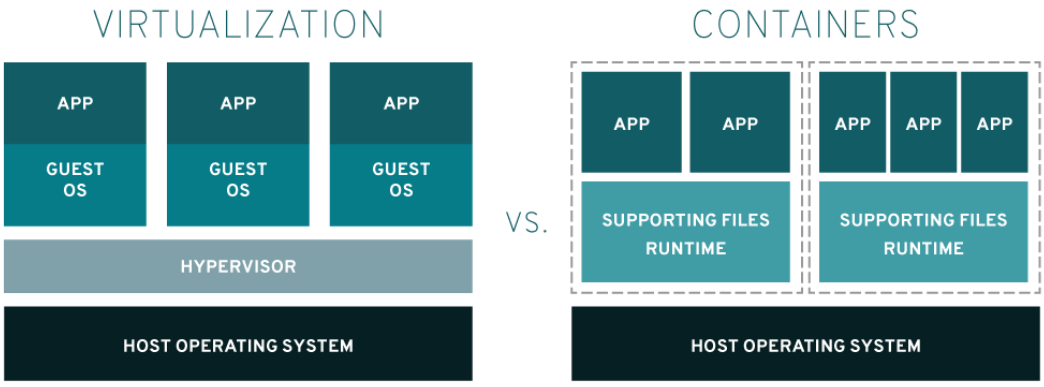
2.2.3 Virtualization

Virtualization sebagai teknologi yang memungkinkan Anda menjalankan beberapa sistem operasi pada satu mesin fisik. Dalam praktiknya, teknologi virtualisasi memungkinkan Anda menjalankan beberapa sistem operasi dan aplikasi pada satu mesin fisik, dengan membagi sumber daya seperti CPU, memori, dan penyimpanan. Dalam *virtualization*, *hypervisor* adalah komponen penting yang memungkinkan virtualisasi terjadi, sesuai pada gambar 2.3. *Hypervisor* adalah *software* yang berjalan pada *physical* server atau mesin dan bertanggung jawab untuk mengelola sumber daya dan memungkinkan beberapa sistem operasi berjalan pada mesin fisik yang sama. *Virtualization* memungkinkan administrator untuk mengelola sumber daya fisik secara terpusat sehingga perangkat keras (*hardware*) bisa dimanfaatkan secara optimal [13].



Gambar 2.3 Manajemen Perangkat Lunak *Virtualization* [13]

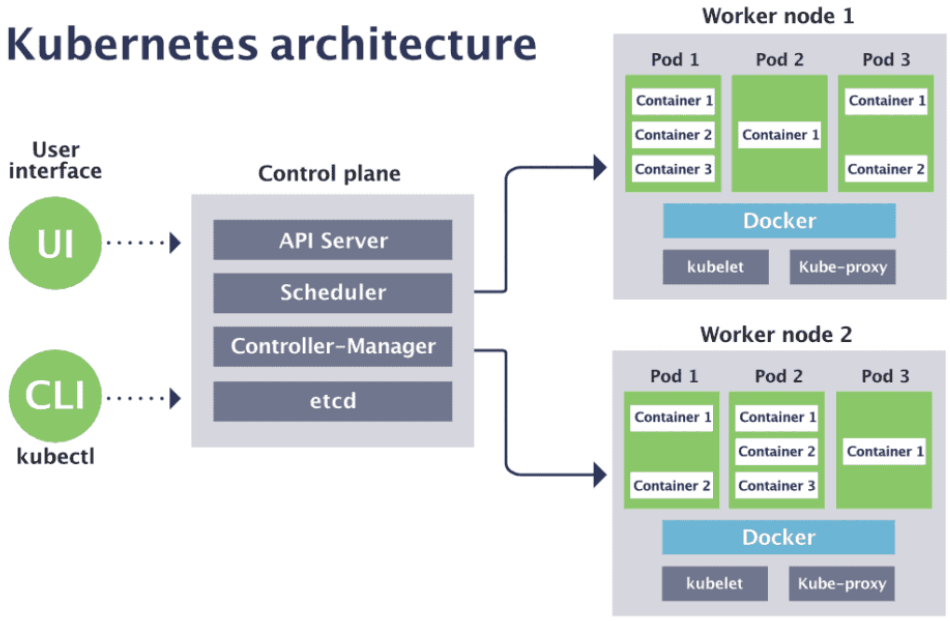
Infrastruktur lama yang sulit dan mahal untuk dipelihara, tetapi masih digunakan untuk aplikasi penting, bisa di-virtualisasikan agar lebih efektif dalam penggunaannya. Perangkat lunak manajemen *virtualization* dirancang untuk memudahkan pengelolaan virtualisasi. Meskipun sebenarnya dapat dilakukan secara manual membagi *resource* menjadi virtual *machine*. *Virtualization* dan *container* berbeda, karena *container* hanya mengisolasi layanan *app* dan *dependency* dari sistem operasi, sedangkan *Virtualization* mengisolasi dari dari level sistem operasi hingga layanan *app* dan *dependency* sesuai pada gambar 2.4 [13].



Gambar 2.4 Perbandingan Arsitektur *Virtualization* dan *Container*[13]

2.2.4 Kubernetes

Kubernetes merupakan suatu yang dikembangkan secara *opensource* untuk mengotomasi penyebaran, penskalaan, dan pengelolaan kontainer. Kubernetes memiliki banyak sekali komponen seperti pada gambar 2.5.

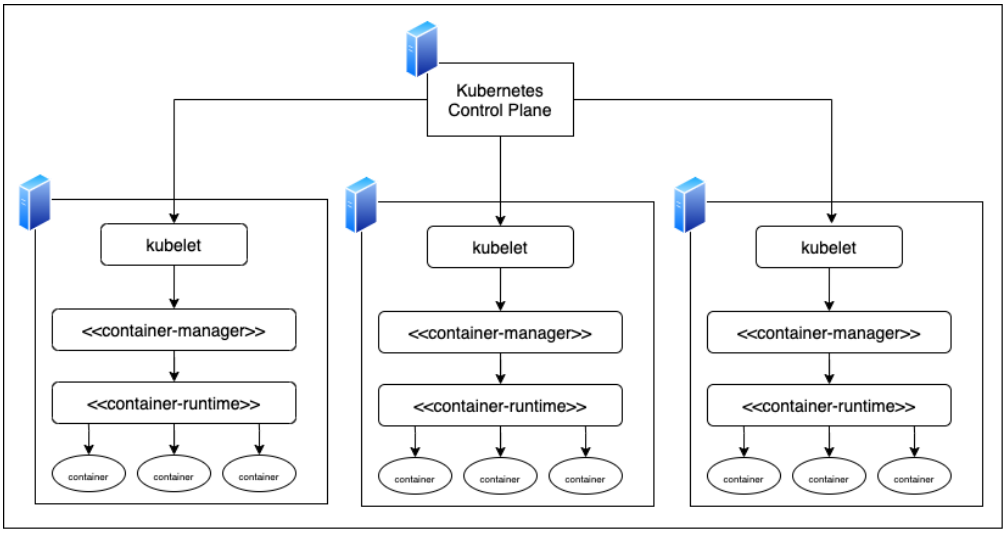


Gambar 2.5 Arsitektur Kubernetes[14]

Gambar 2.5 menunjukkan komponen kubernetes secara keseluruhan. Pada gambar tersebut terlihat ada dua jenis, kubernetes master, dan worker. Kubernetes master menyediakan komponen *control plane* yang mengatur bagaimana pengguna berinteraksi dengan kubernetes. Selain itu komponen master berperan dalam proses pengambilan secara global pada cluster contohnya, mekanisme penjadwalan kontainer, serta berperan dalam proses deteksi serta pemberian respon terhadap peristiwa yang berlangsung di dalam kluster. Pada kubernetes master juga terdapat api server. Komponen ini dapat melakukan pengambilan, pembuatan, memperbarui, dan menghapus sumber daya. Sementara itu, Worker merupakan tempat dimana menjalankan aplikasi yang di *deploy* oleh pengguna. Pada worker terdapat suatu komponen bernama pod. Pod adalah sekelompok satu atau lebih kontainer seperti kontainer containerd, dengan penyimpanan atau jaringan bersama, dan digunakan secara spesifik untuk menjalankan kontainer. Konten Pod selalu ditempatkan bersama dan dijadwalkan Bersama. Pod memodelkan satu atau lebih kontainer aplikasi yang tergabung erat dalam dunia pra-wadah, dieksekusi pada

mesin fisik atau virtual yang sama akan berarti dieksekusi pada host logis yang sama Pod dalam kubernetes dijadwalkan di dalam worker. Pada sebuah kubernetes adalah mesin seperti mesin virtual, server fisik, dll [15].

Selain itu pada worker terdapat komponen yang bernama kubelet. Kubelet adalah *agent* utama yang berjalan pada setiap mesin atau node. Kubelet bertugas mendaftarkan node dengan apiserver pada kubernetes seperti pada gambar 2.6. Kubelet bekerja dalam bentuk PodSpec. PodSpec adalah objek YAML atau JSON yang menjelaskan pod. Kubelet mengambil satu set PodSpec yang disediakan melalui berbagai mekanisme terutama melalui api server dan memastikan bahwa *container* yang dijelaskan dalam PodSpec tersebut berjalan dan sehat. Kubelet tidak mengelola kontainer yang tidak dibuat oleh Kubernetes[16].



Gambar 2.6 Alur Proses Kubernetes *Control Plane* dan Kubelet [17]

2.2.5 Terraform

Terraform adalah *tool* bersifat *opensource* yang dibuat oleh Hashicorp, untuk melakukan penyediaan (*provisioning*) dan pengelolan infrastruktur dengan mendefenisikan pada kode yang dibuat yang kemudian dapat di *deploy* pada infrastruktur (*provider*) yang telah ditentukan[18]. Terraform saat ini memiliki beragam *provider* melakukan *deployment* antara lain layanan *cloud* seperti Microsoft Azure, Amazon Web Service (AWS), Google *Cloud Platfrom* (GCP), Oracle *Cloud* dan masih banyak lagi. Selain itu, terraform menyediakan *provider* untuk KVM Libvirt [19].

2.2.6 Libvirt

Libvirt merupakan sebuah *tool* yang dikembangkan secara *opensource* yang digunakan untuk memanajemen dari teknologi virtualisasi. Libvirt saat ini sudah mendukung untuk berbagai sistem operasi seperti: Linux, Windows dan Mac OS. Libvirt juga mampu mendukung teknologi KVM yangs sebagai *hypervisor*. Untuk menjalankan libvirt dapat menggunakan perintah "*virsh*" [20].

2.2.7 Apache Benchmark

Apache Benchmark adalah sebuah *tool* dari Apache *organization* yang digunakan untuk mengukur performansi pada 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 Bench seperti: *opensource*, *simple command line, load and performance test*. Apache Bench dapat digunakan untuk menguji performa dari web server dengan berbagai batasan pengujian seperti *transfer rate* dan *request per second* [3].

2.2.8 SAR (Sistem Activity Reporter)

Sistem *Activity Reporter* (Sar) adalah program utilitas untuk menganalisis kinerja sistem Linux. Sar merupakan alat pemantauan serba guna yang merupakan bagian dari paket utilitas sumber daya sistem Sysstat. Sar mengukur aktivitas CPU, memory, interrupt, *network trrafic*, I/O, serta *swap*. Data dikumpulkan dalam sistem berkas /proc. Secara default, sar mengumpulkan data sekali setiap 10 menit pada setiap jam [21].

Sar termasuk perintah yang disertakan dalam paket sysstat. Sysstat adalah kumpulan *software* berbasis linux yang digunakan untuk memantau kinerja sistem, paket ini mencakup alat-alat seperti iostat, mpstat, pidstat, sadf, dan sar. Selain outputnya secara *real-time*, sysstat juga akan menginstal sebuah *cronjob* yang akan berjalan setiap 10 menit dan mengumpulkan informasi kinerja sistem. Sar adalah perintah yang dapat Anda gunakan untuk membaca informasi yang telah dikumpulkan tersebut [22].

2.2.9 Throughput

Throughput adalah ukuran kecepatan efektif transfer data, yang diukur dalam bit per detik (bps). Ini mencerminkan jumlah total paket yang berhasil tiba dengan sukses pada tujuan tertentu selama interval waktu yang ditentukan, yang kemudian dibagi oleh durasi interval waktu tersebut. Dengan kata lain, *throughput* menggambarkan seberapa efisien data dapat ditransfer dalam jaringan dengan mengukur jumlah paket yang berhasil dikirim dan diterima selama periode waktu tertentu [23].

Rumus untuk menghitung throughput tercantum pada persamaan 2.1 :

$$Throughput = \frac{\text{Jumlah Paket Yang dikirim}}{Waktu\ Pengiriman\ Data} \quad (2.1)$$

Standar kategori *throughput* tercantum pada tabel 2.2.

Tabel 2.2 Kategori *Throughput*

Kategori	Nilai
Sangat Bagus	>2,1 Mbps
Bagus	1200 Kbps – 2,1 Mbps
Cukup	700-1200 kbps
Kurang Bagus	338-700 kbps
Buruk	0-338 kbps

2.2.10 *Packet loss*

Packet loss merupakan parameter yang menunjukkan jumlah paket yang hilang. Rumus menghitung *packet loss* ditunjukkan pada persamaan 2.3 [23].

$$Packet\ loss = \frac{(\text{paket data dikirim} - \text{paket diterima})\ x\ 100}{paket\ data\ dikrim} \quad (2.2)$$

Standar kategori *packet loss* tercantum pada tabel 2.3.

Tabel 2.3 Kategori *Packet Loss*

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

2.2.11 *Delay*

Delay merupakan waktu yang diperlukan beberapa data untuk mencapai tujuannya di seluruh jaringan atau bisa diartikan dengan waktu yang dibutuhkan informasi untuk mencapai tujuannya dan kembali lagi. Dapat dihitung yang ditunjukkan pada persamaan 2.3 [23].

Rumus untuk menghitung rata-rata *delay* tercantum pada persamaan 2.3 :

$$\text{Rata} - \text{Rata D}elay = \frac{Total\ Delay}{Total\ packet\ yang\ diterima} \quad (2.3)$$

Standar kategori *delay* tercantum pada tabel 2.4.

Tabel 2.4 **Kategori *Delay***

Kategori	Nilai (ms)
Sangat Bagus	< 150
Bagus	150 – 300
Sedang	300 – 450
Buruk	> 400