BAB 2
DASAR TEORI

2.1 KAJIAN PUSTAKA

Penelitian Kusuma, Munadi dan Sanjoyo (2017), penulis melakukan implementasi sistem *clustering* dengan menggunakan *docker swarm* pada *Network File System* untuk menjalankan *Service video on demand*. Hasil dari pengujian tersebut adalah semakin banyak *node worker* yang digunakan , maka performa aplikasi semakin baik, *Latency* pada layanan yang menggunakan 2 *node worker* terjadi penurunan sebesar 70% dibandingkan hanya dengan menggunakan 1 *node* dan saat dilakukan pengujian dengan 3 *node worker* penggunaan CPU juga semakin efisien dengan rata-rata dibawah 50%.[2] Jurnal tersebut mensimulasikan dengan menjalankan layanan *video streaming* dan yang penulis lakukan pada penelitian ini akan melakukan uji coba dengan menggunakan layanan berbasis *web server.* Dan penulis masih sama menggunakan *cluster docker* namun tidak menggunakan *docker swarm* untuk mengatur *cluster docker* secara terdistribusi melainkan *Kubernetes Orchestrator.*

Penelitian Shetty, Upadhya, Rajarajeshwari(2017), Penelitian tersebut menyajikan perbandingan secara rinci antara *Docker container, Openstack* dan *hypervisor* tradisional, Dengan membandingkan dari segi *test* CPU, *Memory* dan disk. Penelitian tersebut menunjukan bagaimana teknologi virtualisasi yang modern yaitu *docker container* sangatlah *powerfull*. Dimana hasil dari perbandingan test *performance* dan *docker container* memiliki benefit yang lebih baik untuk pengembangan aplikasi. *Docker container* juga sangat *lightweight* dan mudah untuk diperluas atau *scale.* Dari penelitian tersebut menginspirasi penulis untuk melakukan penelitian lebih lanjut dengan menggunakan *docker container* namun dengan *orchestrator* dimana akan semakin *powerfull* untuk mengimplementasikan *cluster server* yang menjalankan aplikasi berbasis *web server*[3].

Pada penelitian Sulistyanto (2015), penulis menyatakan *failover cluster* berjalan dengan baik pada *Cluster computer* dengan menggunakan *software Heartbeat,* layanan antar *server* utama dan *server* cadangan (*failover*) tidak jauh

berbeda sehingga *server failover* dapat menjaga kualitas layanan yang disediakan *cluster server. Server cluster* dapat menangani dari segi *availability* pada saat terjadi *failure* pada *server* dari segi kegagalan sistem *hardware.* Nilai dari ketersediaan aplikasi mencapai *availability* paling besar yaitu 99,92% dimana hampir tidak ada masalah *down time*[6] Penelitian ini membahas *high availability* atau tingkat ketersediaan aplikasi dalam *cluster computer* dengan aplikasi *Heartbeat* dan *environment hardware* yang digunakan masih *low end* atau spesifikasi *computer* minimum, yang penulis lakukan dari referensi penelitian ini akan menggunakan *cluster google kubernetes engine* dengan menerapkan pengujian *high availability* yang sama dengan penelitian ini maka penulis juga akan menggunakan *environment* milik *google Cloud platform* dimana server yang disediakan sudah teruji dan siap untuk *production*

Penelitian Satria, Ariefianto dan Yovita (2014), Pengujian *Clustering File System* dengan iSCSI target, *ISCSI Initiator* dan GFS pada SAN, nilai *throughput* dalam jaringan dipengaruhi oleh kualitas *hardware* yang digunakan untuk interkoneksi pada jaringan tersebut[7]. Saat pengujian *failover delay* yang dihasilkan bervariasi antara 6-12 detik yang berpotensi menjadikan adanya *down* time pada sebuah aplikasi dan tidak dapat tersedia bagi pengguna, Dan penulis akan mensimulasikan *fail over* menggunakan *server clustering* untuk *high availability web server* dan *Database Server* dengan *failover delay* yang minim karena adanya peran dari *Kubernetes master* yaitu *kube-controller-manager.*

Gherbi, Kanso dan Li (2015), Menggunakan linux *Container OpenVZ* untuk menerapkan *high availability* pada *service Cloud.* Pengujian menggunakan *service video on demand* untuk menguji performa dari linux *Container* OpenVZ. Penggunaan HA menambah beban CPU *load* namun saat recovery time rata rata hanya dibawah 1 detik yang artinya tidak ada *downtime*[8]. Peneliti menggunakan linux *Container Docker* dan *google kubernetes engine* sebagai *orchestrator* untuk menguji *high availability* pada aplikasi layanan berbasis web.

Pada Penelitian Nettoa, Lunga, Correia dan Luizc (2017), Penelitian tersebut menyajikan pengetesan dari *latency dan throughput* dari *client* yang mengakses *kubernetes cluster* yang telah diterapkan *replication pods* untuk *menghandle request* tertentu dari *client*, penelitian ini menggunakan *Computer*

CPUs *Intel i7* 3.5GHz, *Quad Core cache* L3 8MB, 12GB RAM dan 1TB HDD 7200 RPM, Internet Network 10/100 Mbit dan melakukan penelitian di jaringan lokal dengan menjalankan sistem operasi Ubuntu Server 14.043 64 Bits. Dar hasil paper tersebut menunjukan bahwa mengimplementasikan *kubernetes cluster* untuk melakukan proses permintaan *client* dapat lebih mudah karena banyaknya *pod pod worker* yang dapat *menghandle* seluruh proses permintaan dari *client.* Penulis akan melakukan hal yang sama menggunakan *kubernetes cluster* namun diimplementasi di *environment* milik *google cloud platform* dimana *computer* atau server yang digunakan sudah lebih siap untuk *production*[9].

Tabel 2.1 Rangkuman Keterkaitan dengan Penelitian Sebelumnya.

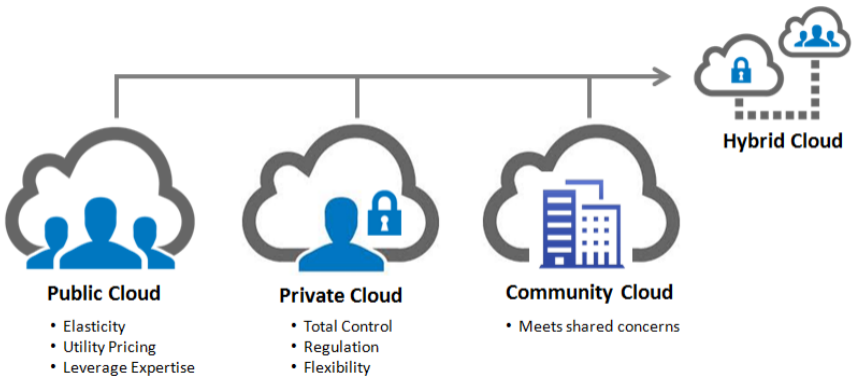
Judul Penelitian	**Penelitian Oleh**	**Tempat Implementasi**			
		***Server* Fisik/Virtual**	**Docker/ LXC**	***Docker Swarm***	***Kubernetes***
Implementasi dan Analisis Computer Clustering System dengan Menggunakan Virtualisasi Docker	Kusuma, Munadi dan Sanjoyo	✓		✓	
An Empirical Performance Evaluation of Docker Container, Openstack Virtual Machine and Bare Metal Server	Shetty, Updhya dan Rajarejeshwari	✓	✓		
IMPLEMENTASI HIGH AVAILABILITY SERVER DENGAN TEKNIK FAILOVER VIRTUAL COMPUTER CLUSTER MAKALAH	Sulistyanto	✓			
IMPLEMENTASI DAN ANALISIS SERVER CLUSTERING MENGGUNAKAN CLUSTER FILE SYSTEM PADA SAN (STORAGE AREA NETWORK) BERBASIS iSCSI UNTUK LAYANAN CLOUD STORAGE	Satria, Ariefianto dan Yovita	✓			
Leveraging Linux Containers to Achieve High Availability for Cloud Services	Gherbi, Kanso dan Li	✓	✓		
State machine replication in containers managed by Kubernetes	Netto, Lunga, Correia dan Luizc	✓			✓
Analisis Performansi High Availability Web Server Pada Cluster GKE (Google Kubernetes Engine) Menggunakan Infrastruktur Google Cloud Platform	Muhammad Naufal Ammar	✓			✓

2.2 DASAR TEORI

2.2.1 *Cloud Computing*

Cloud Penelitian *Computing* merupakan teknologi yang menyediakan sumber daya seperti, *Storage, Operating System*, *Virtual Machine* yang biasa kita kenal VPS *(Virtual Private Server)*. Dengan menggunakan layanan *Cloud Computing* dimana pengguna hanya membayar *services* yang sedang berjalan konsep ini dinamakan *pay-as-you-go* atau basis pembayaran tergantung pemakaian. Perusahaan yang menyediakan layanan tersebut disebut *Cloud Provider*. Beberapa penyedia contoh adalah *Microsoft, Amazon, Google,* dan *Alibaba. Cloud Provider* bertanggung jawab atas perangkat keras yang diperlukan untuk menjalankan aplikasi pengguna[10]. Beberapa macam model *deployment* untuk menerapkan *Cloud computing* yaitu:

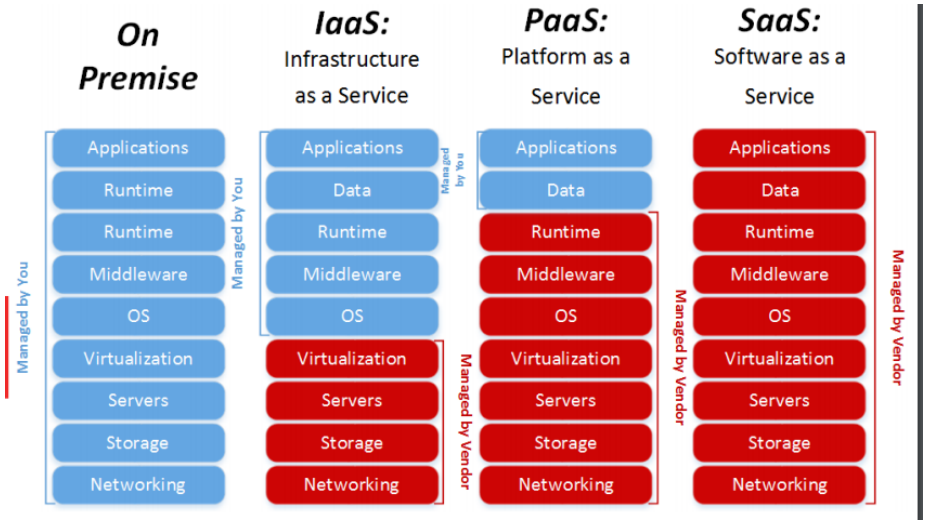
1) *Public Cloud*, Model ini dimana seluruh sumberdaya perangkat keras yang menyediakan adalah *Cloud provider*.
2) *Private Cloud, Cloud* ini dibangun oleh suatu perusahaan atau pebisnis yang ingin menggunakan layanan server untuk menjalankan aplikasi dan bisnis mereka atau biasa disebut *on-premise.*
3) *Hybrid Cloud, Cloud* tipe ini menggabungkan *public Cloud dan private Cloud,* yang memungkinkan pengguna untuk menjalankan aplikasi secara *private* maupun *public* secara tepat. Dan jika menerapkan *hybrid cloud* maka proses *clustering, migration* akan semakin mudah.



Gambar 2.1 *Cloud Deployment models* [11].

Berdasarkan macam macam tipe *Cloud service* ada 3 yaitu :

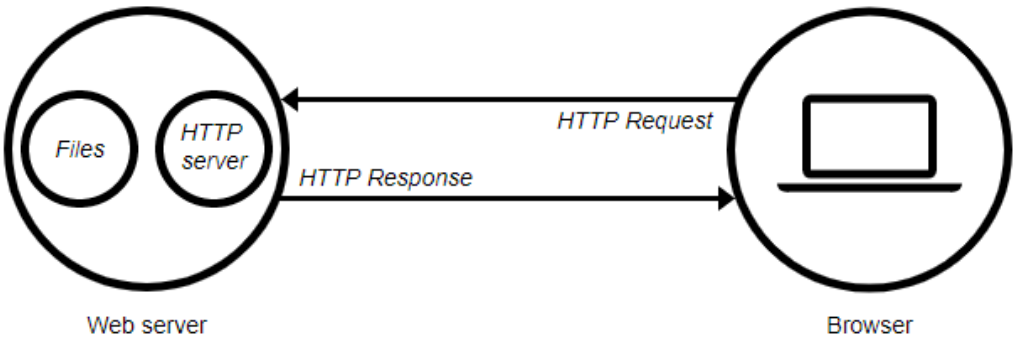
1) *Infrastructure as a service* (IaaS), Layanan ini yang paling fleksibel dan bertujuan untuk memberi bagi pengguna kontrol penuh atas perangkat keras yang menjalankan aplikasi pengguna seperti server infrastruktur mesin virtual (VM), *Storage,* dan *Operating System*, sedangkan *Cloud provider* fokus terhadap ketersediaan teknologi server.
2) *Platform as a service* (PaaS) Menyediakan lingkungan untuk membangun, menguji dan menggunakan aplikasi perangkat lunak. Tujuan PaaS adalah membantu pengguna membuat aplikasi dengan cepat tanpa mengelola infrastruktur yang mendasarinya. Seperti aplikasi web menggunakan PaaS, pengguna tidak perlu menginstall sistem operasi, spesifikasi server yang digunakan seperti apa atau bahkan pembaruan sistem. Jadi pengguna hanya memilih *runtime* apa yang ingin dijalankan kodingan aplikasi web dan berfikir bagaimana infrastruktur yang menjalankan aplikasi web tersebut
3) *Software as a service* (SaaS), SaaS merupakan layanan yang dikelola langsung oleh penyedia, dan pengguna menggunakan layanan aplikasi tersebut tanpa memikirkan infrastruktur aplikasi tersebut. Contoh, *Google Drive, Office 365, Google Docs.*



Gambar 2.2 *Cloud type services* [11].

2.2.2 Web Server

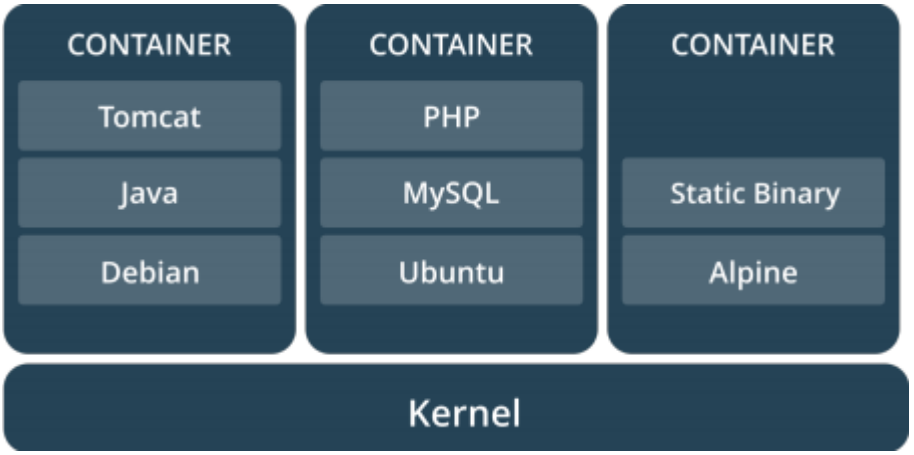
Web server dapat merujuk ke perangkat keras atau perangkat lunak, atau keduanya dapat bekerja sama. Di sisi perangkat keras, *web server* merupakan komputer yang menyimpan file komponen situs web misalnya, Dokumen HTML, gambar, *stylesheet CSS* dan file *Javascript*. Situs web yang berisi file komponen tersebut terhubung ke internet dan mendukung pertukaran data antar *client* dan *server*. Di sisi perangkat lunak, *web server* mencakup beberapa bagian yang mengontrol cara pengguna atau *client* dapat mengakses file web yang ada di *server*, Dengan menggunakan *protokol HTTP/s*. Server HTTP adalah perangkat lunak yang memahami URL (alamat web) dan HTTP (protokol yang digunakan browser *client* untuk melihat halam web). Dan di sisi perangkat lunak kita bisa mengakses *web server* dengan nama domain seperti google.com.[12]



Gambar 2.3 Cara Kerja Web server [11].

2.2.3 Container

Container merupakan aplikasi *open source stand-alone* yang ringan dan berdiri sendiri, dapat dieksekusi yang mencakup semua hal yang diperlukan untuk menjalankan kode, *runtime*, *system tools*, *system library* dan setting. *Container* berjalan dengan cara berbagi sumber daya *kernel* dengan sistem operasi *host*nya[2]. *Container* dapat melakukan pendekatan untuk pengembangan perangkat lunak dimana aplikasi atau layanan, dependensi, dan konfigurasi dikemas bersama menjadi *image container*. Seperti halnya wadah *container* memungkinkan untuk dipindahkan secara mudah dan cepat.

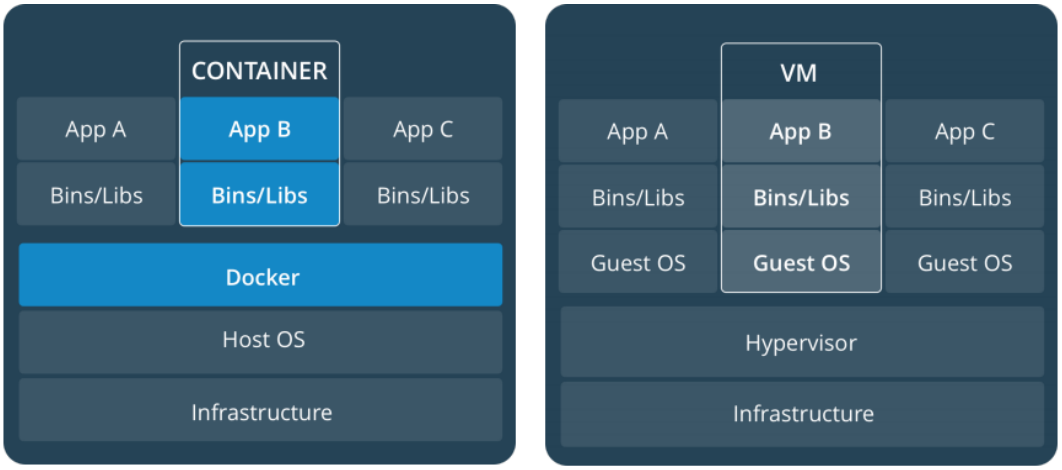


Gambar 2.4 *Layer service containe*r

Container berjalan pada satu mesin dan berbagi *kernel* sistem operasi mesin dan menggunakan lebih sedikit *resource komputasi* dan RAM. *Container* juga beroperasi pada semua distribusi Linux, Microsoft Windows dan setiap infrastruktur termasuk *Virtual Machine*, *Bare-metal Virtualization* dan *Cloud*. *Container* dapat mengisolasi satu aplikasi dengan aplikasi yang lain sehingga dapat menjalankan beberapa aplikasi yang sama.

1) *Container image*, *Container image* merupakan suatu file yang berisi *code executable*, *system library*, *system tools* dan dependensi yang diperlukan untuk menjalankan suatu aplikasi. *Container image* berbagi *kernel* dengan sistem operasi *host*, *Container image* berjalan terisolasi dari *container image* lainnya.
2) Perbedaan *Container* dan *Virtual Machine* , *Container* menjalankan paket kode dan dependensi pada satu layer aplikasi. Beberapa *container* dapat

berjalan pada satu mesin dan berbagi *kernel* sistem operasi dengan *container* yang lain. Setiap *container* berjalan terisolasi satu sama lain. *Container* menggunakan lebih sedikit ruang penyimpanan daripada *Virtual Machine*.



Gambar 2.5 Perbedaan *Layer service containe*r dan *Virtual Machine.*

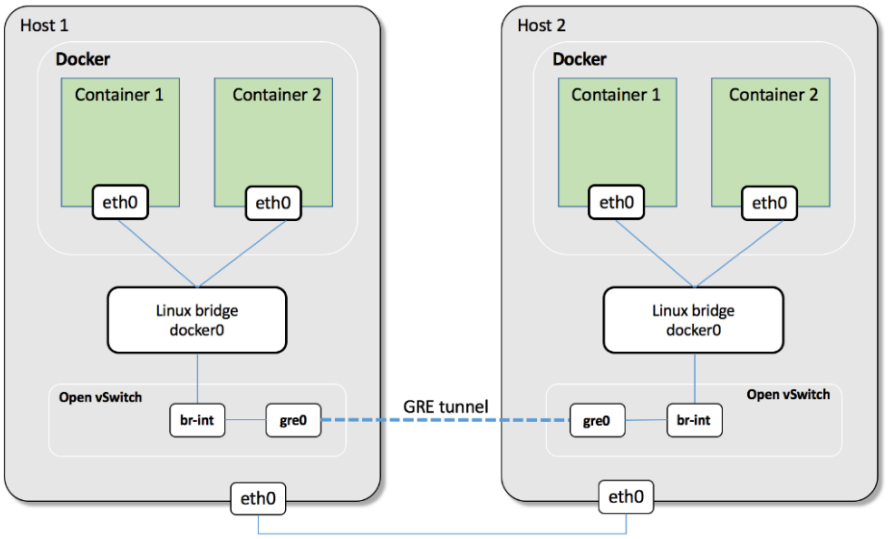
Virtual Machine adalah suatu abstraksi perangkat keras yang mengubah satu server menjadi beberapa server. Satu *Virtual Machine* merupakan satu sistem operasi, beberapa aplikasi yang berjalan di sistem operasi tersebut, *library* dan memakan banyak ruang penyimpanan[7].

3) *Container* Orchestration *Container* orchestrator adalah alat yang digunakan untuk menggabungkan beberapa *host container* menjadi sebuah *cluster*. *Container orchestrator* berfungsi untuk manajemen *container*. Selain itu *container orchestrator* juga berfungsi sebagai *fault-tolerant* untuk mengoptimalkan *service high availability* dari *container*.

2.2.4 *Docker*

Docker merupakan suatu *platform container* yang digunakan untuk mengembangkan sebuah aplikasi yang menjalankan atau membuat aplikasi yang diinginkan. Docker merupakan *application container runtime* yang berjalan *native* berjalan pada sistem operasi Linux, Docker juga dapat berjalan pada sistem operasi Mac OS dan Microsoft Windows[13].

1) *Dockerfile* adalah dokumen teks yang berisi perintah untuk membangun docker *image*. *Dockerfile* berfungsi untuk mengotomatisasi perintah-perintah dalam pembuatan docker *image*. Docker *image* yang telah dibangun kemudian dapat diupload ke *repository* Docker Hub. Docker Hub merupakan repository *registry* resmi untuk mempublikasikan docker *image* yang telah dibangun.
2) Docker *Networking*, *Container* dalam docker menggunakan jaringan *internal* untuk berkomunikasi. Ada beberapa jenis jaringan pada docker yang digunakan untuk berkomunikasi antar *container,* antara lain:
   A. *Bridge*, *Bridge* merupakan *network drivers default* yang digunakan docker. *Bridge* biasa digunakan pada *container* yang berjalan *standalone*.
   B. *Host*, Tipe jaringan *host* menghilangkan isolasi antara *container* dan *host docker*, dan menggunakan jaringan dari *host* secara langsung
   C. *Overlay*, Jaringan *overlay* dapat menghubungkan beberapa *daemon docker* untuk mengaktifkan *service docker* untuk berkomunikasi satu sama lain, Jaringan *Overlay* dapat menghubungkan beberapa *service*.

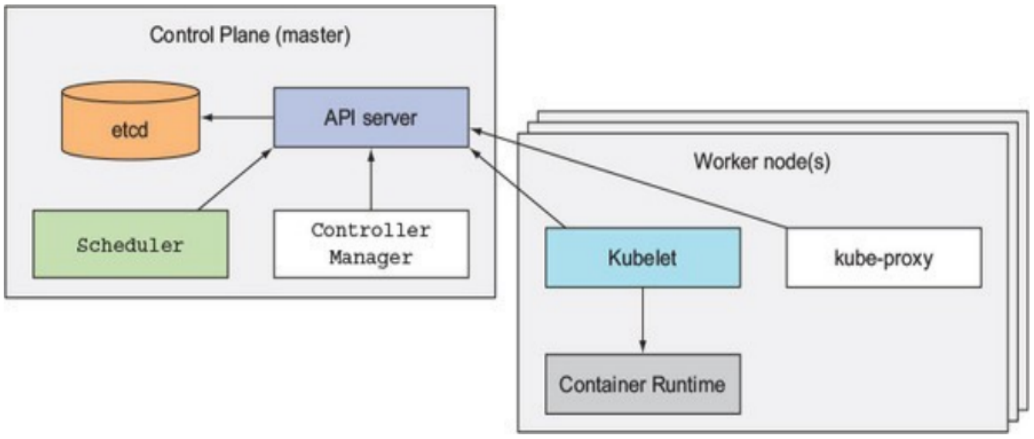


Gambar 2.6 *Overlay network* pada docker [14].

2.2.5 *KUBERNETES*

Kubernetes merupakan *platform cluster open source* yang digunakan untuk otomatisasi dari segi *deployment, scaling* dan manajemen *container. Kubernetes*

merupakan *platform* yang *portable* karena dapat diaplikasikan pada *private* maupun *public* Cloud[8]. Selain itu *kubernetes* hanya menggunakan *resources hardware* yang dibutuhkan sehingga penggunaan *resources* lebih optimal.



Gambar 2.7 *Arsitektur Kubernetes*

Terdapat beberapa komponen di dalam *kubernetes* yaitu komponen *master* dan komponen *node.* Komponen *master* terdiri dari :

1. Kube API server

   Kube API server merupakan *REST* API yang digunakan untuk menangani permintaan dari user. Dengan kube-apiserver user dapat berinteraksi dengan *kubernetes*

2. ETCD

   Etcd digunakan untuk menyimpan data *cluster*. Data konfigurasi dari kbernetes disimpan dalam etcd.

3. Kube *scheduler*

   Fungsi dari *scheduler* adalah untuk menentukan *node worker* mana yang akan digunakan untuk menjalankan suatu *container* baru. *Scheduler* akan memepertimbangkan *resource* yang tersedia untuk mentukan *node* mana yang akan digunakan untuk membuat suatu *pods* baru.

4. Kube *control manager*

   Control manager menangani setiap proses *controller* dalam *kubernetes*, *Controller* merupakan suatu proses yang terpisah, untuk mengurangi

kompleksitas dalam sistem maka *container manager* di *compile* dalam satu *binary* dan berjalan menjadi satu proses.

Komponen *node worker* teridi dari *:*

1. Kubelet

   Kubelet berjalan di setiap *node*. Kubelet melakukan manajemen *pods*, *container*, *image* dan *volume*.

2. Kube *proxy*

   Kube *proxy* berfungsi untuk menyediakan jaringan dalam *node* sehingga *host* dapat menjalankan koneksi *forwarding*. Kube *proxy* berjalan di semua *node*.

3. *Container Runtime*

   Container runtime adalah *software* yang bertugas untuk menjalankan *container*. Pada penelitian ini peneliti menggunakan docker sebagai *container runtime*.

2.2.6 Server *Clustering*

Computer Cluster terdiri dari kumpulan komputer yang saling terhubung dan saling bekerjasama. *Computer Cluster* digunakan untuk meningkatkan kinerja komputasi, karena *cluster* membagi pekerjaan yang sama kedalam beberapa *node* yang berbeda. Dalam *Computer Clustering client* hanya berinteraksi langsung dengan komputer *master,* sehingga *client* hanya dapat melihat satu sistem tunggal. *Server Clustering* adalah menggunakan lebih dari satu *server* yang menyediakan *redundant interconnections,* sehingga *user* hanya mengetahui adanya satu sistem *server* yang diakses. *Clustering digunakan sebagai* redundant *dan backup* sehingga *client* tidak mengetahui apabila terdapat kegagalan pada salah satu *server* dalam *cluster*[1].

2.2.7 *High Availability*

High availability (HA) mempunyai arti akses ke data dan aplikasi kapanpun dibutuhkan dan dengan tingkat kinerja yang dapat diterima. HA berkaitan dengan aspek layanan dari sistem sebagai keseluruhan yang tidak terputus dan dapat dirasakan dari segi *end user* atau *client*[8] . Pada umumnya

arsitektur yang digunakan untuk *High availability* adalah dengan membuat suatu replikasi dari sistem yang sudah diatur dengan *High availability* akan berpindah ke *server* yang sudah di replika dan mengaktifkan replika tersebut. *Client* tidak mengalami gangguan karena dapat ditangani dengan adanya *High availability.*

2.2.8 CPU Usage

CPU *Usage* merupakan presentasi penggunaan CPU pada saat suatu program berjalan. Pengambilan data CPU *usage* bertujuan untuk melihat pembagian beban ke seluruh *node* yang menjalankan *web server* serta mengetahui kinerja dari *web server* [23].

2.2.9 HTTPERF

HTTPERF merupakan aplikasi yang digunakan untuk mengukur kinerja *web server* menggunakan protokol HTTP dengan menawarkan banyak *generator workload*. Fokus utama dari aplikasi ini adalah untuk mengetahui performa *web server* dan memungkinkan analisis performa fitur *server* baru atau perangkat tambahan [19].

2.2.10 Wireshark

Wireshark adalah salah satu perangkat lunak yang berfungsi untuk melakukan monitoring jaringan dengan melakukan *capture,* kemudian hasil *capture* dapat dianalisis untuk mengetahui kinerja jaringan. Wireshark menggunakan *interface* berbasis GUI sehingga banyak dipilih oleh *administrator* jaringan [20].

2.2.11 *QUALITY OF SERVICE* (QOS)

QoS merupakan suatu metode pengukuran yang digunakan untuk mengetahui kemampuan suatu jaringan agar dapat memberikan layanan *network* yang lebih baik dan terencana [21]. QoS mendefinisikan karakter dan sifat dari suatu *service* sehingga dapat mengukur beberapa atribut kinerja yang telah dispesifikasikan dengan suatu *service.*