BAB II
TINJAUAN PUSTAKA

2.1 Tinjauan Pustaka

Penelitian mengenai analisis *High Availability web server* pada docker sudah banyak dilakukan menggunakan metode yang sesuai dengan permasalahannya masing-masing. Berikut merupakan penelitian terdahulu yang sudah pernah dilakukan :

Pada penelitian pertama berjudul "*Analisis High Availability Web Sever Pada Cluster Docker Menggunakan Container Orchestrator Kubernetes Dengan Simulator GNS3(Graphical Network Simulator 3)* ditulis pada tahun 2019 oleh Yusuf Al-afid. Penggunaan internet yang terus bertambah diseluruh dunia membuat penyediaan layanan aplikasi web harus memikirkan ketersediaan *resource* untuk *web server* yang dikelola. Satu server tidak bisa menangani proses yang sangat banyak dan akan membebani kinerja *web server*. Diperlukan suatu sistem yang dapat membantu kinerja dan *availability server*. Berdasarkan latar belakang tersebut, solusi yang dapat diberikan dengan menyediakan *High Availability* yang tinggi pada server dengan menggunakan teknologi *server clustering*. Dengan menggunakan kubernetes salah satu *container orchestrator* yang berfungsi untuk manajemen *container*. Hasil dari penelitian ini adalah didapatkannya nilai *high availability* yang mampu dicapai oleh kubernetes dan pemulihan dalam sistem.

Pada penelitian kedua "*Implementasi High Availability Cluster Web Server Menggunakan Virtualisasi Container Docker"* ditulis pada tahun 2019 oleh Muhammad Aldi Aditia Putra. Meningkatnya permintaan akan kebutuhan informasi dalam internet menyebabkan jalannya beban trafik pada *web server* semakin banyak. Banyak permintaan yang masuk menuju layanan *web server* dapat menyebabkan server menjadi *overload* karena tidak dapat melayani semua permintaan. Untuk meringankan beban yang berlebihan dalam suatu server diterapkanlah konsep *clustering*, dimana server bekerja bersama sama seperti satu sistem tunggal. Solusi untuk masalah tersebut dengan penerapan *Load Balancing* pada *cluster web server* menggunakan virtualisasi *container* docker akan menjadi lebih efisien. Penerapan *Load Balancing* pada *cluster web server* diharapkan dapat

menjadi solusi dalam mengatasi permintaan layanan yang banyak kedalam *web server* sehingga tidak terjadinya *overload*. Hasil dari penelitian ini adalah kemampuan sistem dengan *load balancing* dengan dua buah *web server cluster* menggunakan algoritma *least-connection* lebih baik dalam melayani *request client* dengan nilai *request* per-detik 2607.141 req/s.

Pada penelitian ketiga berjudul " *Implementasi Load Balancing Server Web Berbasis Docker Swarm Berdasarkan Penggunaan Sumber Daya Memory Host"* ditulis pada tahun 2019 oleh Mohamad Rexa Mei Bella. Virtualisasi bebasis container sangat popular dikalangan *programming development* di karenakan virtualisasi yang ringan dimana kernel Linux dapat membagi penggunakan *resource* antar container bertujuan agar kinerja container tidak saling terganggu antar lainnya. Salah satu virtualiasi berbasis container yang sering digunakan adalah Docker. Dalam mengatur *container* dalam jumlah yang banyak sangatlah rumit, tetapi docker memiliki *engine* untuk mengaturnya yang disebut docker swarm. Docker swarm memiliki *load balancing* internal tetapi hanya mengatur antar *container* dalam *host* dan tidak dapat di monitoring. *Load balancing* dapat mengoptimalkan suatu web server dengan membagi beban *traffic* dengan berbagai algoritma dan metode yang berbeda. Hasil dari penelitian ini adalah mendeteksi *node worker* yang memiliki beban sedikit dalam proses *request* sehingga dapat didistribusikan dan dapat mendeteksi *host* saat *down* ketika bekerja.

Tabel 2. 1 Penelitian Terdahulu

No	Judul	*Comparing*	*Contrasting*	*Criticize*	*Synthesize*	*Summarize*
1	*Analisis High A vailability Web Sever Pada Cluster Docker Menggunakan Container Orchestrator Kubernetes Dengan Simulator GNS3(Graphical Network Simulator 3)*	Melakukan penelitian terkait kinerja server dalam menyediakan layanan dengan *clustering server* docker swarm. Selain itu, penelitian yusuf(2019) menggunakan kubernetes.	Membahas ketersediaan dalam menyediakan layanan terhadap *client* menggunakan kubernetes, sedangkan penulis juga membuat ketersediaan dengan menggunakan docker swarm	Tidak ada perbandinga n dengan *orchestrator* yang lain.	Penulis menyamakan beban kinerja yang diberikan kepada server agar dapat membandingkan dengan penelitian terdahulu	Penelitian menunjukan bahwa penggunaan *clustering server* dalam memberikan layanan terhadap *client* mampu menyediakan ketersediaan sebesar 99,97%. Semakin banyak *nine* maka semakin tinggi sebuah sistem *availability*nya

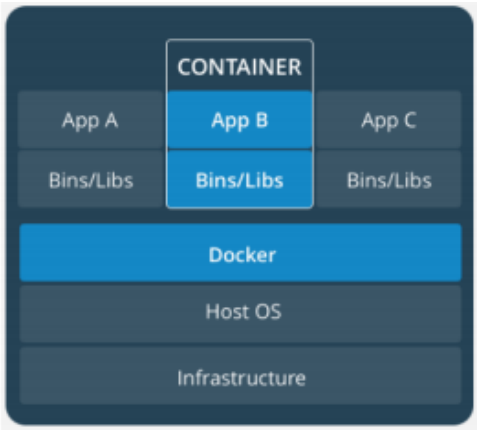
No	Judul	*Comparing*	*Contrasting*	*Criticize*	*Synthesize*	*Summarize*
2	*Implementasi High Availability Cluster Web Server Menggunakan Virtualisasi Container Docker*	Melakukan penelitian terkait kinerja server dalam menyediakan layanan dengan *clustering server* docker swarm. Selain itu, penelitian aldi(2020) dilakukan untuk mencari nilai *request* per-detik dan *throughput* menggunakan *clustering server*	Membahas perbandingan penggunaan *single server* dengan *dua cluster server* dan membandingkan penggunaan algoritma *round robin* dan *least connection*, sedangkan penulis hanya mencari nilai *high availability* dan nilai *throughput*	*High availability* kurang dalam pembahasan dan tidak adanya pembahasan tentang aplikasi *web* yang digunakan	Penulis mengintegrasikan 2 isu penting yang ditulis dalam penelitan aldi(2020) agar dapat memperdalam penelitian. 2 isu tersebut adalah *throughput* dan CPU *utilization*.	Penelitian menunjukan bahwa penggunaan dua *cluster server* dengan menggunakan algoritma *least connection* dapat menghasilkan nilai *request* 2607.141 per-detik

No	Judul	*Comparing*	*Contrasting*	*Criticize*	*Synthesize*	*Summarize*
		dengan docker swarm.				
3	*Implementasi Load Balancing Server Web Berbasis Docker Swarm Berdasarkan Penggunaan Sumber Daya Memory Host*	Melakukan penelitian terkait kinerja server dalam menyediakan layanan dengan *clustering server* docker swarm. Selain itu, penelitian Rexa(2019) melakukan implementasi *load balancing* untuk mengurangi *traffic* web server.	Membahas kemampuan *load balancing* pada docker swarm dalam membagi beban *traffic* berdasarkan penggunaan *memory*, sedangkan penulis tidak membahas penggunaan sumber daya	Hanya menggunakan 1 parameter dalam kinerja *load balancer*.	Penulis mengintegrasikan 2 isu penting yang ditulisdalam penelitan Rexa(2019) agar dapat memperdalam penelitian. 2 isu tersebut adalah *load balancing* dan penggunaan sumber daya.	Hasil penelitian menunjukan arsitektur dapat diterapkan sesuai dengan yang diharapkan dan *load balancing* berdasarkan penggunaan sumber daya *memory* dapat mengurangi *traffic* web server.

2.2 Landasan Teori

2.2.1 *Container*

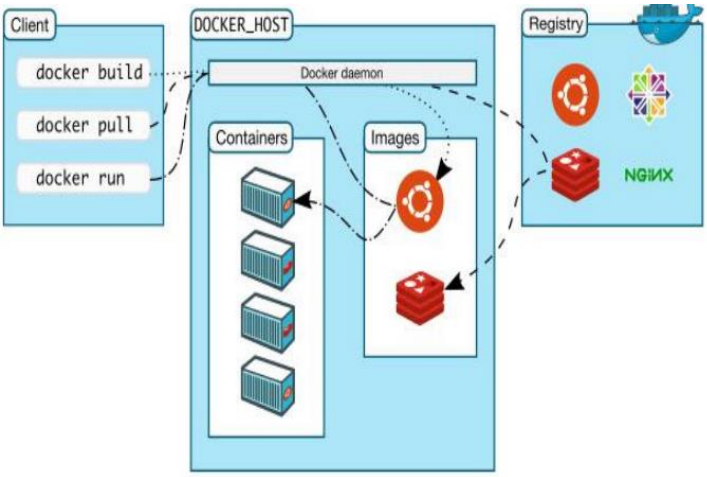
Container adalah paket perangkat lunak ringan yang dapat berdiri sendiri, yang mencakup semua yang dibutuhkan untuk menjalankannya : *code, runtime*, alat sistem, *library* sistem, dan pengaturan[6]. *Container* berjalan dengan cara berbagi sumber daya *kernel* dengan sistem operasi *hostnya*. Oleh sebab itu *Container* memiliki ukuran file yang lebih kecil, karena *container* tidak perlu menyiapkan sistem operasi secara penuh. Seperti yang terlihat pada gambar 2.1



Gambar 2. 1 Cara Kerja *Container*

2.2.2 *Docker*

Docker adalah suatu *platform* terbuka bagi pengembang perangkat lunak dan pengelola sistem jaringan untuk membangun, mengirimkan dan menjalankan aplikasi-aplikasi terdistribusi[7]. Docker tidak hanya dapat dijalankan dalam sistem operasi *Linux* tetapi docker juga dapat berjalan pada sistem operasi *Microsoft* dan *Mac OS*.



Gambar 2. 2 Arsitektur Docker

2.2.2.1 *Docker Images*

Docker images yaitu tatanan yang hanya bisa melakukan *read-only* untuk menjalankan *containers*. *Image* dapat terdiri dari sistem operasi dan beberapa aplikasi yang sudah terpasang, *image* yang berada dilapisan paling bawah disebut dengan *base image* sedangkan *image* yang berada dilapisan paling atas disebut dengan *parent image*.

2.2.2.2 *Docker Hub*

Docker Hub atau biasa disebut juga *Docker registry* komponen ini berfungsi sebagai tempat penyimpanan *docker images* di *cloud base*. Selain berfungsi sebagai tempat penyimpanan docker hub menyediakan ribuan *docker images*, saat *user* ingin memasang aplikasi dapat melakukan perintah *pull* untuk mendownload dan menjalankannya secara langsung. *User* juga dapat melakukan perintah *push* melalui docker *client* ke *docker registry* untuk penyimpanan maupun *sharing*.

2.2.2.3 *Docker File*

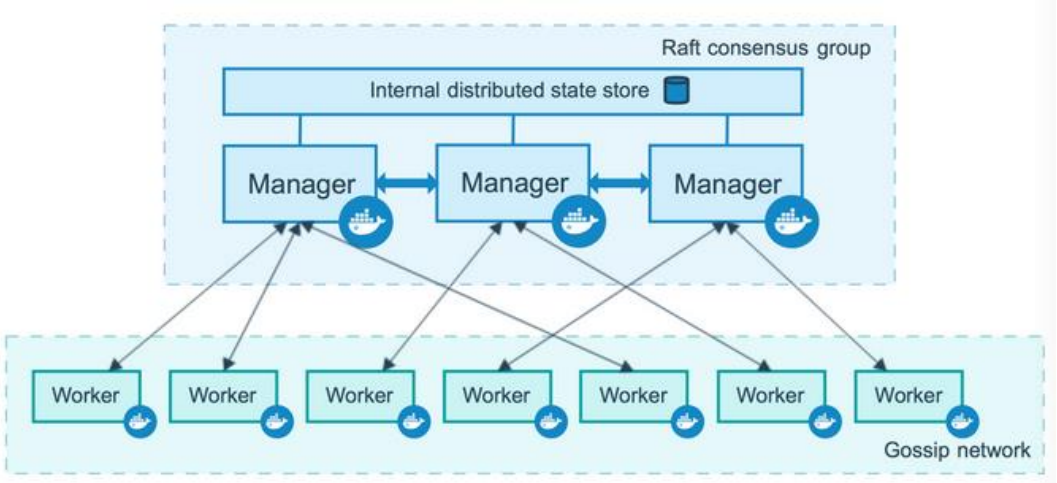
Docker File yaitu suatu file dengan format penulisan *.yaml* yang dipergunakan oleh *docker engine* untuk menciptakan suatu *image* berdasarkan baris perintah dan konfigurasi yang terdapat pada file dan melalui *docker-commit*.

2.2.2.4 *Docker Compose*

Docker Compose merupakan alat bantu tambahan yang sangat membantu dan mempermudah dalam proses pengembangan maupun pengimplementasian suatu *service*, Docker compose dapat menjalankan beberapa *docker container*, dengan menggunakan file *compose* untuk mengonfigurasi layanan aplikasi. Dengan menggunakan satu perintah dapat membuat dan memulai semua layanan dari konfigurasi yang sudah dibuat pada *docker file* dengan format penulisan *yaml*.

2.2.3 Docker Swarm

Docker swarm adalah *docker native clustering solution*, yang dapat mengubah sekolompok *host* docker terdistribusi menjadi satu *server* virtual besar[6]. Swarm terdiri dari beberapa *host* yang berjalan dalam mode *swarm cluster*, dalam swarm suatu host dapat bertindak menjadi manager yang berfungsi untuk mengelola sumber daya, sedangkan *host* lainnya bertindak sebagai *worker* yang menjalankan layanan swarm.



Gambar 2. 3 Ilustrasi Arsitektur Docker Swarm

2.2.4 Server Clustering

Kumpulan komputer yang saling terhubung dan saling bekerjasama disebut juga *Computer Cluster*. *Cluster Server* merupakan teknologi yang menggabungkan beberapa sumber daya yang bekerja bersama-sama sehingga tampak seolah-olah merupakan suatu sistem tunggal[8]. Penggunaan *clustering* pada server berguna untuk melakukan *backup* jika salah satu *server* terjadi kegagalan sehingga *client*

tidak akan mengetahui. Pembentukan *server clustering* dibutuhkan setidaknya dua server, terlebih dahulu mengatur autentikasi yang digunakan dalam *cluster* barulah menambahkan *node* kedalam *cluster* yang telah dibuat dengan mendefinisikan *host* dan *password* yang sudah dibuat dan aktifkan *cluster* yang sudah dibuat. Penggunaan teknologi *cluster* tidak hanya digunakan untuk *server* saja contoh lain yang bisa menerapkan teknologi *cluster* yaitu *Virtual Machine Cluster* merupakan teknologi yang efektif dalam memastikan ketersediaan tinggi pada server dan jaringan. VM *cluster* menggunakan mesin virtual sebagai *node*, sehingga bekerja dengan melindungi mesin fisik dari segala kegagalan perangkat keras dan lunak. *Cluster Physical and Virtual Machine* dimana setiap mesin fisik dan mesin virtual masing-masing membuat *cluster*. Setiap mesin virtual menjalankan perangkat lunak pengelompokan. Kegagalan pada perangkat keras pada mesin fisik, mesin virtual *host* siaga dapat mengambil alih untuk *host* fisik tersebut.

2.2.5 *High Availability*

High Availability adalah ketersediaan yang tinggi dari *web server* secara terus menerus untuk melayani kebutuhan pengguna (*client)* dalam sistem tersebut[2]. Dalam memberikan aspek layanan yang dilakukan secara terus menerus, pengguna (*client*) tidak akan merasakan gangguan meskipun terdapat salah satu server yang mengalami kegagalan. *High Availability* pada umumnya menggunakan dengan membuat replika pada *web server* dan *database server.* Saat terjadinya kegagalan pada salah satu server maka *server* lain yang menjalankan replika tersebut.

Pengukuran *availability* memiliki istilah ”*nine*”, semakin banyak “*nine*” maka semakin tinggi sebuah sistem *availability*[4].

Perhitungan *availability* berdasarkan rumus yang digunakan oleh Braastad dan Calzolari seperti berikut.

$$\textbf{Availability} = \frac{\textbf{MTBF}}{\textbf{MTBF+MTTR}} \text{ x } 100$$

- *Mean time Between Faults* (MTBF)
  Rata-rata *uptime*
- *Mean time to Repair* (MTTR)
  Rata-rata waktu yang diperlukan dalam memulihkan layanan