BAB I
PENDAHULUAN

1.1 Latar Belakang

Dalam era digital yang semakin berkembang pesat, *hosting aplikasi web* telah menjadi elemen penting dalam menyediakan layanan dan konten kepada pengguna di seluruh dunia. *Hosting* aplikasi *web* melibatkan menyimpan dan mengelola aplikasi serta data terkait di *Server* yang dapat diakses melalui *internet*. Sebagai teknologi berkembang, sistem jaringan dan masalah manajemen sumber daya yang terbatas muncul. Sistem teknis sangat dibatasi oleh ruang seperti penawaran *Server*, penawaran penyimpanan, dan peningkatan kapasitas perangkat keras. Tentu saja biaya awal ini tidak murah. Selain itu, *Server* tidak dapat digunakan secara optimal sebagai perangkat keras dasar. *Server* adalah pusat kendali otoritas untuk masalah jaringan. *Server* sering tidak bekerja secara optimal karena beban dan permintaan yang tinggi. Oleh karena itu, *Server* hanya menggunakan semua sumber daya untuk permintaan tertentu. Situs *web* dengan *traffic data* yang tinggi dapat menyebabkan beban kerja yang berat di sisi *Server*, yang pada gilirannya akan mengakibatkan turunnya perfomansi *Server*, bahkan kegagalan sistem secara keseluruhan. Salah satu solusi yang dapat dilakukan untuk mengatasi masalah performa *web Server* dalam hubungannya dengan jumlah *Request* yang meningkat adalah pemutakhiran perangkat keras *web Server* namun solusi ini hanya bersifat sementara(Nurzaman dkk., 2022).

Di sisi lain, teknologi virtualisasi kontainer menjadi sebuah topik hangat di dunia IT, dikarenakan virtualisasi yang ringan dimana dapat menyediakan fitur portabilitas bagi aplikasi perangkat lunak sehingga dapat berjalan pada sebagian besar *platform*, tetapi juga memiliki kekurangan dimana penggunaan teknologi ini mungkin melibatkan kurva belajar dan kompleksitas tambahan dalam manajemen dan pemeliharaannya, sehingga penggunaan teknologi ini menjadi suatu tantangan tersendiri. Salah satu virtualisasi berbasis kontainer yang paling banyak digunakan

saat ini adalah Docker. Docker adalah salah satu platform yang dibangun berdasarkan teknologi kontainer. Docker merupakan sebuah *project open-source* yang menyediakan *platform* terbuka untuk *developer* maupun *sysadmin* untuk dapat membangun, mengemas, dan menjalankan aplikasi dimanapun sebagai sebuah wadah (kontainer) yang ringan. (Rexa dkk., 2019)

Docker memiliki salah satu produk yaitu *Docker Swarm* untuk mendeploy kontainer pada *multihost*. *Docker Swarm* adalah *container orchestration* dan *clustering tool* untuk mengatur *Kontainer hosts*, yang merupakan bagian dari *Docker engine*. *Docker swarm* merupakan *original clustering* yang dibuat dan disediakan oleh *Kontainer* sehingga menjamin *high availability* dan *high performance* aplikasi yang berjalan. Kelebihan *Docker Swarm* adalah jika salah satu *host down* maka *service Kontainer* akan digantikan dengan *host* yang sedang aktif. *Docker Swarm* memiliki dua *node* yang bekerja, yaitu *node Manager* dan *node Worker*. *Node Manager* adalah *node* yang mengatur dan memberikan tugas(*task*) kepada *node Work*, sedangkan *node worker* adalah node yang bekerja sesuai dengan tugas yang diberikan oleh *node Manager*. Mekanisme penyeimbang bebab internal *Docker Swarm* berfokus pada bagaimana mendistribusikan permintaan pada *node worker* secara seimbang berdasarkan permintaan dari pengguna. (Putri dkk., 2021)

Menurut (Khalida dkk., 2019) menjelaskan bahwa *Docker container* sangat cocok untuk desain arsitektur sistem dengan pendekatan *microservice,* karena masing-masing service memiliki lingkungan yang terisolasi namun tetap dapat berkomunikasi satu sama lain. *Microservice* sendiri merupakan suatu pendekatan desain arsitektur sistem informasi yang menspesifikasikan dan memecah fungsi dari sistem yang besar menjadi sistem atau *service* yang kecil dan spesifik. Adapun saran untuk penelitian selanjutnya adalah perlu banyak referensi tentang konfigurasi *web Server* desain dan pengembangan aplikasi berbasis *Container* ini karena pada proses pengembangan aplikasi hal ini dapat mempengaruhi desain arsitektur suatu sistem. Di dalam penelitian ini terdapat kekurangan dimana belum menerapkan virtualisasi berbasis *Docker Swarm* yang mana belum menjamin aspek *high availability* dan *high performance* aplikasi yang berjalan.

Menurut (Rexadkk.,2019)penggunaan *load balancing* dapat mengoptimalkan suatu *web Server* dengan membagi beban *traffic* dengan bebagai algoritma dan metode yang berbeda. Oleh karena itu peneliti menggunakan metode *load balancing* berbasis sumber daya Metode ini dipilih karena dengan mengetahui penggunaan *memory web Server* kita bisa mengetahui *Nodeworker* mana yang bebannya sedikit memproses suatu *Request* sehingga beban dapat di distribusikan dengan baik dan dapat mendeteksi *host* yang *down* ketika bekerja. Penelitian ini memaparkan tentang *load balancing* pada kontainer *Nginx*. Kekurangan dari penggunaan *load balancing* pada kontainer *Nginx* adalah hanya membagi *Request client* antar kontainer sehingga dapat mengakibatkan penggunaan *resource* yang tidak merata antar mesin *host* dan tidak dapat memonitor *resource* yang digunakan. Salah satu *resource* yang penting dalam mesin *host* adalah *memory*.

Menurut (Wibawa dkk., 2023.) penggunaan *load balancing* dalam *Kontainer* dinilai lebih stabil dikarenakan *respone time* lebih rendah, penggunaan *CPU* yang lebih tinggi, dan penggunaan *memory* yang lebih efisien. Sehingga algoritma *round robin* lebih baik dalam *load balancing web Server* di *Docker Swarm*. Di dalam penelitian ini terdapat kekurangan dimana belum membandingkan algoritma yang ada pada *load balancing*.

Berdasarkan pada kesimpulan penelitian yang telah dilakukan oleh Khalida dkk, Rexa dkk dan Wibawa dkk. Penggunaan *load balancing* masih menggunakan metode *round robin* dimana metode ini akan membagikan beban *Request* yang masuk pada *load balancing* secara merata kepada *Server* lain. Kekurangan dari yaitu metode ini tidak menghiraukan kondisi *Server* saat dikirimi beban *Request* oleh *load balancing*. Penelitian sebelumnya juga belum membandingkan penggunaan *environtment* lain yang digunakan untuk menguji performansi optimal suatu *web*site. Mengacu pada uraian diatas, maka tujuan dari penelitian ini adalah untuk membandingkan performansi dari *web*site yang dipasang langsung pada sever lokal, *Server* virtual berupa aplikasi di dalam kontainer dan *Server* terdistribusi yang dipasangi *load balancing* serta menguji performa *web*site jika tidak dipasang *load balancing* sebagai salah satu solusi dalam meningkatkan kinerja dari sebuah *web*site. Sehingga diharapkan dapat menjadi pertimbangan

dalam proses operasional *Web Server* secara efektif dan memiliki performa yang baik dan perbandingannya menghasilkan persentase nilai dengan selisih yang signifikan.

1.2 Rumusan Masalah

Rumusan masalah dalam penelitian ini adalah bagaimana perbandingan kinerja antara *Server web* tradisional, *Server web* berbasis kontainer, dan *Server web* terdistribusi dalam menangani permintaan pengguna dan bagaimana respon waktu, *throughput*, dan persentase *error* berbeda di antara ketiga jenis *Server web* ini dalam situasi penggunaan yang berbeda.

1.3 Batasan Masalah

Batasan masalah dari Penelitian ini akan membatasi analisis pada *Server web* yang menggunakan teknologi dan platform tertentu, dan tidak akan mencakup *Server* aplikasi atau sistem lainnya serta Kinerja *Server web* akan diukur berdasarkan respon waktu, *throughput*, dan persentase *error*, tanpa memasukkan faktor lain seperti keamanan.

1.4 Tujuan

Tujuan yang ingin dicapai dari penelitian ini adalah :

1. Untuk membandingkan dan mengevaluasi efisiensi perbedaan kinerja antara *server web* tradisional, *Server web* berbasis kontainer, dan *Server web* terdistribusi dalam menangani permintaan pengguna.
2. Menentukan *Server web* mana yang memiliki kinerja terbaik dalam menangani *client Request* dalam berbagai kondisi penggunaan dan beban kerja.

1.5 Manfaat

Penelitian ini memiliki manfaat yaitu membantu memastikan bahwa beban kerja di *web server* didistribusikan secara merata di antara *server-server* yang ada serta dengan menggunakan algoritma *load balancing* yang tepat, setiap *Server* akan menerima jumlah permintaan yang seimbang, menghindari *overload* pada beberapa *Server* dan *underutilization* pada *Server* lainnya.