Analisis Perbandingan *Performasi Respon Waktu Web Server* dan *Failover* Antara Kubernetes Dan Docker Swarm pada Container Orchestration

Sugiyatno[1], Ishak M[2]

[1,2] Program Studi Informatika, STMIK EL RAHMA Yogyakarta

e-mail: [1]sugiyatno@stmikelrahma.ac.id [2]ishaqm@gmail.com

Abstrak

Failover sebagai mekanisme dalam mengatasi single failure terintegrasi pada kubernetes dan docker swarm sebagai server cluster berbasis container atau orkestrasi container. Namun, perbedaan performansi failover dan respon waktu web server pada kubernetes dan docker swarm belum diketahui.

Berangkat dari permasalahan tersebut, diperlukan suatu analisis yang mendalam terkait perbandingan performansi failover dan respon waktu web server antara kubernetes dan docker swarm dengan melakukan berbagai pengujian dan percobaan yang dilakukan pada masing-masing cluster tersebut.

Hasil pengujian dengan menggunakan percobaan upload file untuk mengetahui performansi respon waktu web server pada masing-masing cluster, ditemukan data bahwa respon waktu kubernetes 54,5% lebih cepat dibandingkan dengan docker swarm. Lalu pada pengujian failover dari sisi node failure, docker swarm memiliki waktu rata-rata failover yang sangat signifikan. Namun, docker swarm tidak memiliki management container seperti management pods yang terdapat pada kubernetes, sedangkan pada saat pengujian failover ketika proses upload file berlangsung, menampilkan bahwa kedua cluster memiliki hasil yang sama yaitu website mengalami error ketika pengujian dilakukan.

Kata kunci*— container orchestration, kubernetes, docker swarm, failover*

1. PENDAHULUAN

Jaringan dapat dikatakan sebagai sebuah hubungan maupun komunikasi yang dilakukan oleh beberapa perangkat yang terhubung pada sebuah titik tertentu. Beberapa jaringan juga dapat dihubungkan maupun disatukan dengan menggunakan sebuah teknik yang disebut virtualisasi. Virtualisasi merupakan sebuah teknik yang dimana dapat menciptakan sebuah wadah virtual dari sebuah objek yang memiliki wujud fisik seperti wadah penyimpanan data, sistem operasi dan juga *server*. Salah satu teknik virtualisasi yaitu virtualisasi berbasis kontainer.

Virtualisasi berbasis kontainer merupakan sebuah teknik dimana virtualiasi tersebut tidak memerlukan *hypervisor* untuk berfungsi, kontainer ini dapat dijalankan langsung pada sistem operasi Pada *VMWare* mengharuskan pengguna untuk mengatur *resource* ketika instalasi, sedangkan pada kontainer tidak diharuskan karena kontainer memiliki *resource* tersendiri yang disediakan oleh *host*. Jika kekurangan *resource* maka kontainer akan mengambil *resource* yang terdapat pada *hardware* sesuai yang dibutuhkan. Salah satu virtualisasi berbasis kontainer yaitu Docker.

Docker merupakan sebuah tempat atau wadah yang menggunakan sistem berbasis kontainer, yang digunakan dalam mengembangkan aplikasi *web server* maupun *mobile apps* guna mempermudah fase *deploy* pada *software*.[1] Definisi tersebut membawa pengertian praktis bahwa Docker merupakan suatu cara memasukkan layanan ke dalam lingkungan terisolasi bernama container, sehingga layanan tersebut dapat dipaketkan menjadi satu bersama dengan semua pustaka dan software lain yang dibutuhkan.[2]

Namun, di era teknologi yang semakin maju dimana aplikasi *web* ataupun *mobile* menjadi kebutuhan dasar untuk mendapatkan informasi maupun berbagi data. Sebuah aplikasi *web* dan *mobile* tentu membutuhkan sebuah *server* untuk pengembangan aplikasi. Namun, hal tersebut tidak akan cukup jika hanya menggunakan satu *server hosting*, karena kegagalan *server* dalam

merespon permintaan atau *single point of failure* dapat terjadi kapan saja. Akibat dari kegagalan server tersebut dapat membuat aplikasi tidak dapat diakses oleh klien.

Server cluster merupakan sebuah gabungan dari beberapa komputer *server* yang digunakan oleh sebuah lembaga maupun organisasi untuk mencapai kebutuhan yang diperlukan oleh server untuk melampaui kemampuan sebuah mesin.[1] Pergantian lalu lintas jaringan dari koneksi utama ke koneksi cadangan akan berjalan secara otomatis. Failover dapat memudahkan administrator jaringan dalam manajemen jaringan, karena tidak perlu melakukan konfigurasi jaringan jika jaringan utama down. Dengan adanya failover juga dapat menjamin ketersediaan tinggi dan jaringan yang andal.[3]

Server juga bertugas untuk menjalankan software administratif yakni software yang mengontrol akses terhadap jaringan dan sumber daya yang terdapat di dalamnya. Hal ini termasuk file atau alat pencetak (printer), dan memberikan akses kepada workstation anggota jaringan. Di dalam sistem operasi server, umumnya terdapat berbagai macam service yang menggunakan arsitektur klien/server. Contoh dari service yang diberikan oleh server ini antara lain Mail Server, DHCP Server, HTTP Server, DNS Server, FTP Server dan lain lain. Setiap sistem operasi server umumnya merangkai berbagai layanan tersebut. Atau bisa juga layanan tersebut diperoleh dari pihak ketiga. Setiap layanan tersebut akan merespons terhadap request dari klien.[4]

Failover pada umumnya diimplementasikan untuk tujuan meningkatkan ketersediaan layanan yang disediakan. Elemen klaster akan bekerja dengan node-node redundan yang kemudian digunakan untuk menyediakan layanan saat salah satu elemen klaster mengalami kegagalan. Ukuran yang paling umum dari kategori ini adalah dua node, yang merupakan syarat minimum untuk melakukan redundansi. Implementasi klaster jenis ini akan mencoba untuk menggunakan redundansi komponen klaster untuk menghilangkan kegagalan di satu titik (*Single Point of Failure*).[5]

Failover merupakan salah satu fitur yang terdapat pada banyak platform termasuk Kubernetes dan Docker Swarm. Kubernetes dan Docker Swarm merupakan sebuah *platform open source* berpusat pada pengelolaan kontainer yang salah satu fungsinya adalah membuat sebuah sistem *server cluster* yang menjadi wadah kerja sama bagi beberapa server dalam penyediaan layanan. Teknologi ini menjadi solusi permasalahan penurunan performansi server database yang bersifat sentralisasi dan dengan adanya teknologi virtualisasi ini administrator dapat membuat/mempunyai server database cadangan sebagai backup apabila server database utama nya *down*.[6]

Container adalah unit *software* yang mengemas semua kode dengan dependensinya, sehingga aplikasi dapat berjalan dengan cepat dari satu lingkungan komputasi ke lingkungan komputasi yang lain. Sehingga, dengan kata lain container dapat diasumsikan sebagai suatu wadah yang dapat menampung berbagai data dan aplikasi yang bertujuan untik membuatnya menjadi lebih ringkas dan efisien untuk berjalan pada sistem.[7]

Berdasarkan permasalahan beserta konsep penyelesaian yang dijabarkan di atas, Kubernetes dan Docker swarm memberikan suatu solusi dalam mengatasi *server failure* dengan mekanisme *failover*. Penelitian ini dilaksanakan dengan mengimplementasikan Kubernetes dan Docker swarm guna untuk menganalisis mekanisme dan respon *failover* dalam mengatasi *server failure* serta menguji performansi respon waktu web server ketika file upload yang berjalan didalam cluster.

2. METODE PENELITIAN

2.1. Analisa Masalah

Berdasarkan latar belakang yang telah diuraikan, maka rumusan masalah pada penelitian ini adalah:

a. Perbandingan infrastruktur dan mekanisme failover pada Kubernetes dan Docker Swarm.

b. Belum diketahui berapa lama waktu yang dibutuhkan ketika proses failover berlangsung pada orkestrasi Kubernetes dan Docker Swarm.
c. Perbandingan waktu respon website antara Kubernetes dan Docker Swarm

2.2. Alat dan Bahan

a. Alat

1. Macbook Pro (Spesifikasi Ram 8GB)
2. Lenovo Ideapad 3 (Spesifikasi Ram 8GB)

b. Bahan

1. Terminal Iterm
2. Docker
3. Kubernetes
4. Sublime Text
5. Vagrant
6. GitBash
7. Docker Image OS Ubuntu Server

2.3. Metode Pengumpulan Data

Metode pengumpulan data yang dapat digunakan adalah:

a. Metode Observasi

Metode observasi adalah metode pengumupulan data dengan melakukan pengamatan langsung terhadap obyek yang diteliti untuk mengumpulkan data dan informasi yang berkaitan dengan permasalahan yang ada.

b. Metode Studi Literatur

Studi literatur merupakan teknik pengumpulan data yang dilakukan dengan cara mengumpulkan dan membaca berbagai sumber tertulis seperti buku atau literatur yang menjelaskan tentang landasan teori. Pengumpulan data dan informasi dilakukan melalui penggalian informasi dan pengetahuan dari sumber-sumber seperti buku, karya tulis, serta beberapa sumber lainnya yang ada hubungannya dengan objek yang relevan terhadap penelitian.

2.4. Langkah-Langkah Penelitian

Langkah-langkah yang dilakukan dalam penelitian ini adalah seperti pada Gambar 1.

a. Perencanaan

Tahap awal yang dilakukan dalam proses perencanaan yaitu kegiatan ini dilakukan setelah pengumpulan data yang diperlukan untuk membuat sistem dan hasil dari pengumpulan data adalah sebagai bahan dalam perancangan sistem.

b. Perancangan dan Desain

Setelah melakukan pelaksanaan penelitian dan kajian literatur sehingga didapatkan data digital, maka selanjutnya dilakukan perancangan sistem yaitu persiapan mesin, desain topologi jaringan dari server cluster dan perancangan mekanisme yang digunakan.

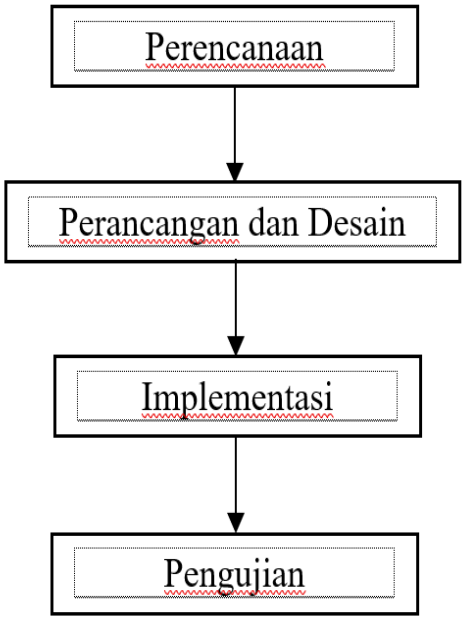
c. Implementasi

Implementasi dari perancangan dan desain sistem yang digunakan yaitu dengan instalasi server cluster dan pemasangan software dan tools dari platform yang sudah ditentukan.

d. Pengujian

Pengujian sistem dilakukan untuk mengetahui tingkat kesalahan dan keberhasilan sistem. Proses uji coba ini dilakukan untuk memastikan sistem yang dibuat sudah benar

dan sesuai dengan karakteristik yang telah diterapkan serta tidak ada kesalahan di dalamnya.



Gambar 1. Tahapan penelitian

3. HASIL DAN PEMBAHASAN

3.1. PERANCANGAN SISTEM

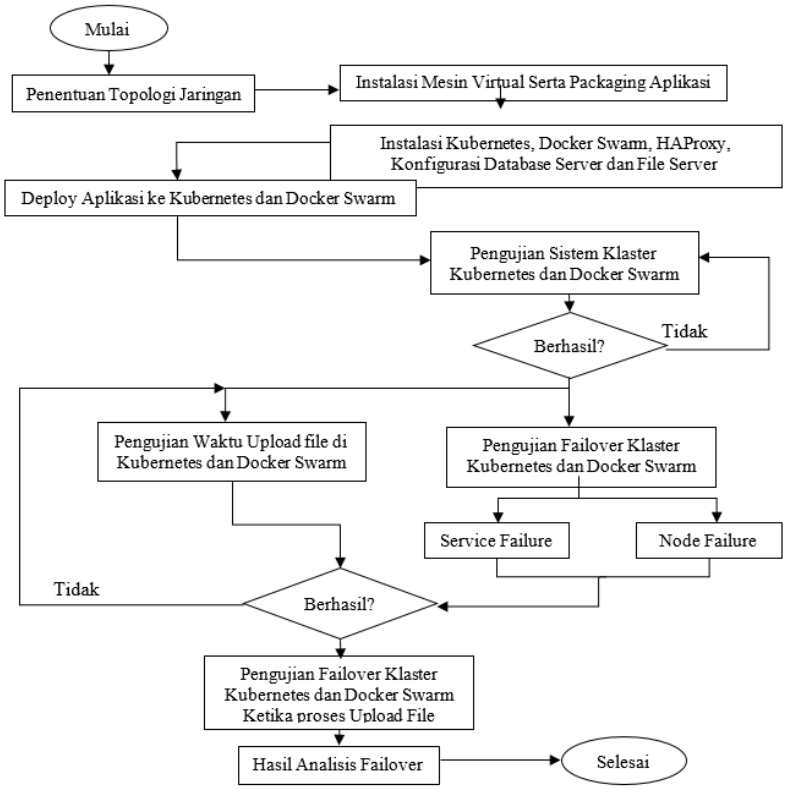
Perancangan Sistem yang bertujuan untuk mendeskripsikan tentang alur penelitian, diagram infrastruktur, topologi jaringan yang digunakan dan instalasi paket yang dibutuhkan. Sehingga bisa memberikan Gambaran secara rinci penelitian ini.

Analisis penelitian dilakukan mulai dari memasukkan proses pembuatan topologi jaringan sampai dengan pengambilan data hasil analisis dari waktu respon Ketika *upload file, failover klaster* dan *failover* Ketika proses *upload file.*

Pada Gambar 2 terdapat diagram alur perancangan sistem agar hasil penelitian dapat tercapai. Diagram tersebut dimulai dari penentuan topologi jaringan sebagai dasar untuk menjalankan penelitian yang di dalamnya berisi penentuan mesin virtual yang digunakan, teknologi yang digunakan, pengaturan alamat IP, pembagian *resource* pada masing-masing mesin virtual, sistem operasi dan aplikasi yang digunakan.

Jika topologi telah ditentukan, selanjutnya pemasangan sistem operasi yang dalam penelitian ini adalah Ubuntu Server 20.04 dan aplikasi yang telah melalui proses *packaging* ke dalam Docker *image container* yang akan digunakan lalu *push* atau mengunggah docker images ke *Docker Registry*. Lalu pada proses selanjutnya yaitu instalasi Kubernetes Cluster, Docker Swarm beserta dependensi dan *setup* HAProxy.

Jika proses sebelumnya berhasil, maka langkah berikutnya adalah melakukan proses *deploying* ke Docker Swarm dan Kubernetes Cluster. Apabila proses *deploying* telah selesai, maka langkah selanjutnya adalah mengerjakan pengujian pada sistem Kubernetes Cluster dan Docker Swarm untuk mengetahui aplikasi yang melewati tahap *deploying* sudah dapat diakses tanpa kendala.



Gambar 2. Diagram Alur Penelitian

Lalu pada pada pengujian kedua adalah pengujian *failover* yang berfungsi untuk mengetahui waktu yang dibutuhkan kubernetes cluster dan docker swarm ketika terjadi *failover*. Mekanisme failover pada penelitian ini berfokus pada pengujian *service failure* dan *node failure*. *Service failure* adalah suatu kegagalan yang terjadi pada aplikasi, *pods* atau *service*, sedangkan *node failure* merupakan kegagalan yang terjadi pada *node worker*, mesin atau *server* pada *cluster*.

Proses pengujian selanjutnya yang dilakukan ketika dua pengujian sebelumnya telah dilakukan adalah pengujian *failover* ketika proses *upload file* sedang berlangsung. Pengujian ini dilakukan untuk mengetahui kendala-kendala apa saja yang mungkin ditemukan dalam melakukan *upload file* atau terdapat perbedaan waktu respon ketika mesin yang memproses *website* sedang *down* lalu terjadi *failover*.

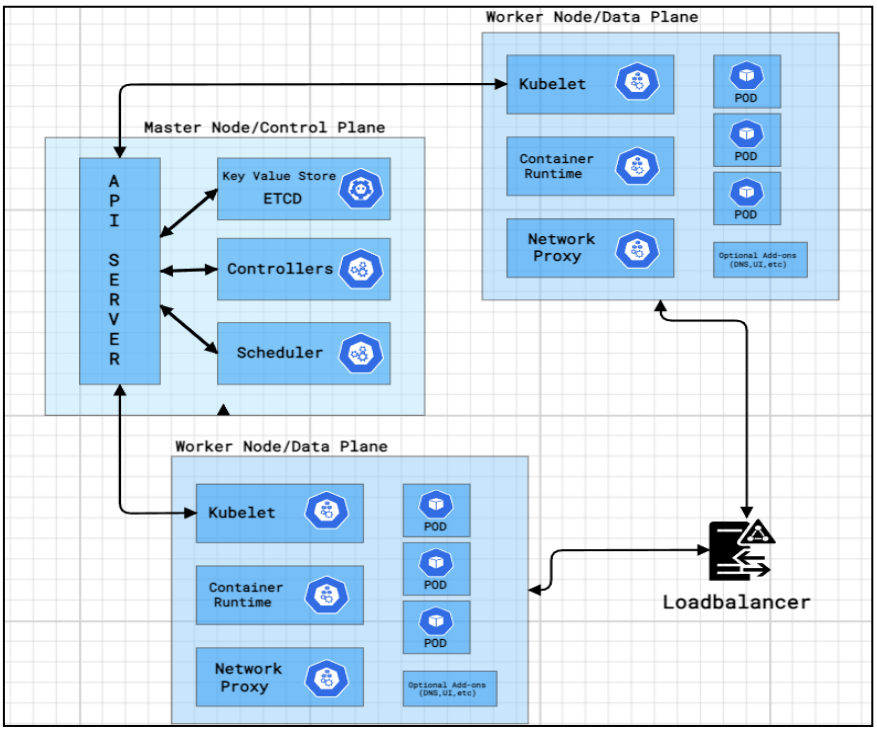
Usai melakukan pengujian, tahap selanjutnya adalah pengambilan data dari hasil pengujian yang telah dilakukan. Parameter yang diambil adalah durasi waktu ketika *upload file*, saat *failover* dari *service* dan *node* berlangsung serta *failover* dalam tahap *upload file* berlangsung pada masing-masing *Container Orchestration* yang digunakan.

3.2. Diagram Infrastruktur Kubernetes Cluster

Deskripsi pada Gambar 4.3 di bawah menunjukkan bahwa dalam pengelolaan *cluster*, administrator dari Kubernetes Cluster membutuhkan akses langsung ke dalam *node master (control plane)*, melalui CLI ataupun UI (*dashboard*) agar dapat berkomunikasi langsung dengan API Server yang membutuhkan *tool* yaitu kubectl.

API Server merupakan komponen Kubernetes yang berinteraksi langsung dengan administrator. Etcd bertugas untuk menyimpan seluruh konfigurasi dan status *cluster* dengan parameter status *node, pod* dan *container*. *Controller* bekerja untuk memeriksa keadaan dan status *node* dan *scheduler* berperan dalam pengawasan permintaan baru dari API server.

Jika terdapat permintaan untuk menyebarkan *pod* ke *node worker*, maka *pod* tersebut akan dimasukkan ke *scheduler* terlebih dahulu untuk diproses dan dijadwalkan sebelum ke *node* yang berada dalam *cluster*.



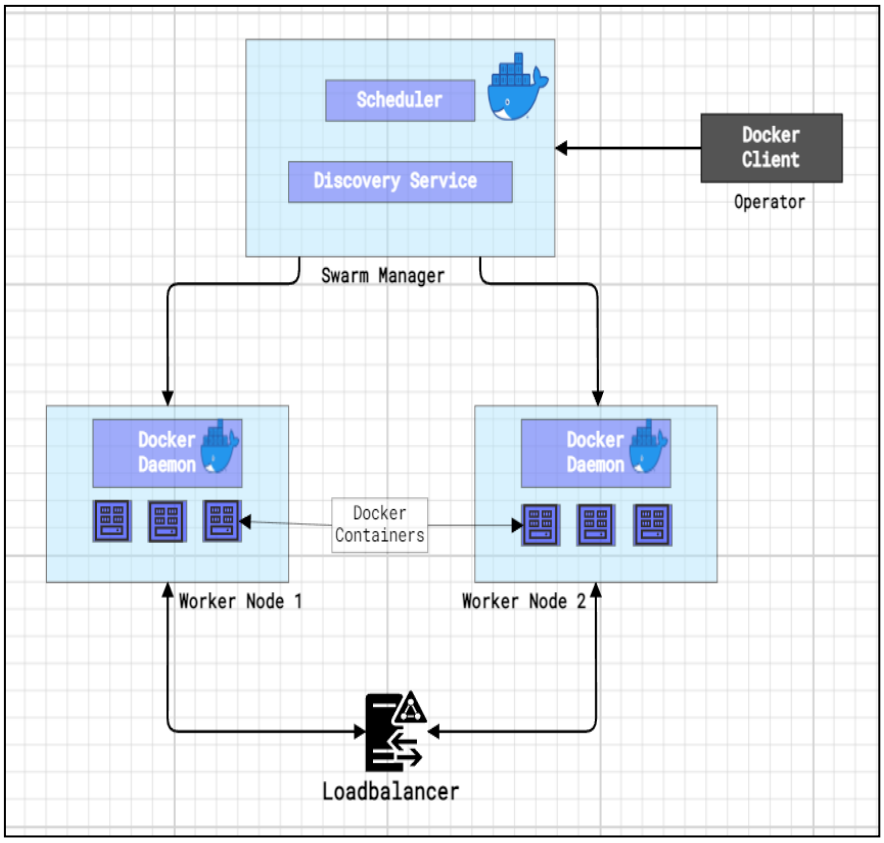
Gambar 3. Diagram Infrastruktur *Kubernetes Cluster*

Deskripsi pada Gambar 3 di atas menunjukkan bahwa API Server merupakan komponen kubernetes yang berinteraksi langsung dengan administrator. Etcd bertugas untuk menyimpan seluruh konfigurasi dan status *cluster* dengan parameter status *node, pod* dan *container*. *Controller* bekerja untuk memeriksa keadaan dan status *node, scheduler* berperan dalam pengawasan permintaan baru dari API server.

Fungsi dari *node worker* atau *data plane* adalah bertugas untuk melayani permintaan akses aplikasi dari *end user*. Node worker berkomunikasi langsung dengan API Server menggunakan Kubelet yang merupakan agen utama dari kubernetes yang bertugas mengawasi dan melaporkan sumber daya seperti *CPU, RAM dan storage*, serta mengawasi dan melaporkan kondisi pod ke node master.

Tahapan berikutnya adalah melakukan *runtime* pada *container* yang bertugas mengelolah *image container* agar kubernetes dapat berkomunikasi dengan *registry image container*. Kube-proxy bertujuan untuk memastikan bahwa setiap *node* mendapatkan alamat IP-nya. Implementasi *iptables* dan aturan lokal berfungsi untuk menangani proses pembuatan rute dan penyeimbangan beban lalu lintas.

Pod sebagai unit terkecil dalam kubernetes cluster merupakan unit paling penting karena aplikasi yang disebarkan tersebut akan dibungkus (*packaging*) ke *dalam* pod tersebut. Lalu pada *loadbalancer* bertugas sebagai *front-end network* dari suatu infrastruktur atau merupakan mesin yang harus diakses oleh pengguna.



Gambar 4. Diagram Infrastruktur Docker Swarm

Deskripsi pada Gambar 4 di atas menunjukkan bahwa dalam pengelolaan *cluster*, administrator akan melakukan akses langsung ke dalam *Swarm Manager (control plane)* melalui CLI agar dapat berkomunikasi langsung dengan *cluster*. Pada *cluster* ini terdapat 3 *node*, yaitu *node manager*, *node worker1* dan *node worker2*. *Node manager* dengan fitur *scheduler* bertugas untuk melakukan penjadwalan dalam mengelola cluster dan *discovery service* adalah mekanisme yang digunakan docker untuk merutekan permintaan dari klien ke *node cluster*, tanpa klien perlu mengetahui berapa banyak *node* yang berpartisipasi dalam layanan atau alamat IP atau *port*.

Lalu fungsi pada *docker daemon* untuk memproses pengelolaan docker *images*, *container*, *network*, dan *storage volumes*. Docker Daemon menerima request dari docker API dan akan memprosesnya. Docker API disini merupakan komponen yang digunakan untuk berinteraksi dengan *docker daemon*. Komponen ini bisa diakses klien melalui HTTP.

Docker Container adalah wadah yang berfungsi untuk mengemas dan menjalankan aplikasi. Wadah ini mencakup kode, *runtime*, *system tools*, dan pengaturan. Container hanya bisa mengakses resource yang telah ditentukan dalam *docker image*. Lalu pada *loadbalancer* bertugas sebagai *front-end network* dari suatu infrastruktur atau merupakan mesin yang harus diakses oleh pengguna.

3.3. Pengujian Respon Waktu Upload File

Pada tahap pengujian ini, dilakukan beberapa kali percobaan untuk menemukan durasi yang diperlukan saat pengiriman data pada masing-masing kluster. Percobaan yang dilaksanakan melalui prosedur *upload file* sebanyak 30 kali dengan data yang telah ditentukan yaitu *file* Gambar berukuran 10MB.

Tabel 1. *Upload File* pada Kubernetes Cluster

Data Upload	Mulai Upload	Selesai Upload	Lama Upload(detik)
1	00:17:51	00:18:03	00:00:12
2	00:19:31	00:19:41	00:00:10
3	00:20:16	00:20:26	00:00:10
4	00:21:27	00:21:38	00:00:11
5	00:22:38	00:22:51	00:00:13
6	00:23:41	00:23:52	00:00:11
7	00:24:22	00:24:33	00:00:11
8	00:25:40	00:25:51	00:00:11
9	00:26:37	00:26:50	00:00:13
10	00:27:33	00:27:44	00:00:11
11	00:28:19	00:28:32	00:00:13
12	00:29:02	00:29:15	00:00:13
13	00:29:52	00:30:02	00:00:10
14	00:30:34	00:30:44	00:00:10
15	00:31:26	00:31:36	00:00:10
16	00:32:08	00:32:19	00:00:11
17	00:32:58	00:33:10	00:00:12
18	00:33:49	00:34:02	00:00:13
19	00:34:43	00:34:58	00:00:15
20	00:35:40	00:35:50	00:00:10
21	00:36:15	00:36:26	00:00:11
22	00:37:31	00:37:42	00:00:11
23	00:38:21	00:38:31	00:00:10
24	00:39:34	00:39:46	00:00:12
25	00:40:26	00:40:36	00:00:10
26	00:41:06	00:41:18	00:00:12
27	00:41:45	00:41:56	00:00:11
28	00:42:38	00:42:49	00:00:11
29	00:43:17	00:43:28	00:00:11
30	00:43:56	00:44:06	00:00:10
Rata-rata			**00:00:11**

Tabel 2. Upload File Pada Docker Swarm

Data Upload	Waktu Mulai	Waktu Selesai	Lama Upload(detik)
1	10:30:07 PM	10:30:31 PM	00:00:24
2	10:32:03 PM	10:32:24 PM	00:00:21
3	10:33:46 PM	10:34:11 PM	00:00:25
4	10:36:55 PM	10:37:23 PM	00:00:28
5	10:38:01 PM	10:38:23 PM	00:00:22
6	10:38:57 PM	10:39:18 PM	00:00:21
7	10:40:10 PM	10:40:36 PM	00:00:26
8	10:41:27 PM	10:41:50 PM	00:00:23

9	10:42:22 PM	10:42:40 PM	00:00:18
10	10:43:16 PM	10:43:35 PM	00:00:19
11	10:44:16 PM	10:44:36 PM	00:00:20
12	10:45:09 PM	10:45:34 PM	00:00:25
13	10:46:09 PM	10:46:29 PM	00:00:20
14	10:47:18 PM	10:47:34 PM	00:00:16
15	10:48:35 PM	10:48:53 PM	00:00:18
16	10:49:30 PM	10:49:51 PM	00:00:21
17	10:50:28 PM	10:50:48 PM	00:00:20
18	10:51:22 PM	10:51:42 PM	00:00:20
19	10:52:13 PM	10:52:32 PM	00:00:19
20	10:53:05 PM	10:53:22 PM	00:00:17
21	10:54:12 PM	10:54:28 PM	00:00:16
22	10:55:10 PM	10:55:29 PM	00:00:19
23	10:56:00 PM	10:56:20 PM	00:00:20
24	10:57:00 PM	10:57:19 PM	00:00:19
25	10:57:44 PM	10:58:02 PM	00:00:18
26	10:58:34 PM	10:58:51 PM	00:00:17
27	10:59:22 PM	10:59:41 PM	00:00:19
28	11:00:14 PM	11:00:38 PM	00:00:24
29	11:01:08 PM	11:01:25 PM	00:00:17
30	11:01:51 PM	11:02:12 PM	00:00:21
Rata-rata			**00:00:20**

3.4. Pengujian Failover

Prosedur pengujian bertujuan untuk mendapatkan rata-rata waktu yang dibutuhkan dan mengetahui model mekanisme *failover* dari Kubernetes dan Docker Swarm. Pengujian dilakukan dengan menggunakan dua skenario yaitu *service failure* dan *node failure*.

Tahap pengujian dilakukan melalui proses pengamatan terhadap *web server* yang berjalan pada *cluster*. Pengamatan berfungsi untuk mengetahui konsistensi kinerja dari *web server* yang dapat tetap berjalan pada saat terjadi kegagalan atau mengalami kendala yang dapat berakibat pada berhentinya pengoperasian *web server*.

Tabel 3. Status *Node Worker Kubernetes Cluster*

Status		**Akses**
Node Worker 1	***Node Worker 2***	
Menyala	Menyala	*Up*
Menyala	Tidak Menyala	*Up*
Tidak Menyala	Menyala	*Up*

Tabel 4. Status *Node Worker Docker Swarm*

Status			**Akses**
Node Manager	***Node Worker 1***	***Node Worker 2***	

Menyala	Menyala	Menyala	*Up*
Menyala	Menyala	Tidak Menyala	*Up*
Menyala	Tidak Menyala	Menyala	*Up*
Menyala	Tidak Menyala	Tidak Menyala	*Up*
Tidak Menyala	Menyala	Menyala	*Up*

4. KESIMPULAN

Dari proses percobaan, pengujian dan analisis yang telah dilakukan, ditemukan beberapa data bahwa performansi respon waktu web server dan failover pada kubernetes dan docker swarm memiliki perbedaan yang cukup signifikan.

1. Proses pengiriman file pada kubernetes lebih cepat dibandingkan docker swarm dengan perbedaan rata-rata waktu 11 detik pada kubernetes dan 20 detik pada docker swarm. Dari hasil rata-rata waktu yang ditemukan, persentase perbedaan kecepatan pengiriman file dari kedua cluster tersebut adalah 54,5%.
2. Mekanisme failover dan respon waktu yang dimiliki kubernetes cenderung lebih lambat dan lama dibandingkan docker swarm dengan perbedaan rata-rata waktu yang cukup signifikan yaitu 345 detik atau setara dengan 5,45 menit pada kubernetes dan 21 detik pada docker swarm. Hal tersebut sangat dipengaruhi oleh infrastruktur pada docker swarm yang sudah terintegrasi dengan docker engine sedangkan kubernetes menjadikan docker engine sebagai komponen pihak ketiga dan ketika failover terjadi, scheduler kubernetes perlu melewati rules dan kebijakan yang dapat menyebabkan keterlambatan dalam mengalihkan workload ketika failover.
3. Kubernetes memiliki fitur untuk mengelola *pods/container* dan *service network* untuk mengelola jaringan yang dapat diatur sedemikian rupa berdasarkan kebutuhan sedangkan docker swarm tidak memiliki fitur untuk mengelola *container* dan *service network,* namun hal tersebut dapat dilakukan dengan perintah dasar docker.
4. Pada pengujian failover ketika upload file sedang berlangsung pada masing-masing cluster, tidak ditemukan perbedaan. Pengujian tersebut sama-sama memberikan hasil yaitu web server gagal merespon atau koneksi terputus ketika merespon permintaan atau memproses pengunggahan tersebut. Kegagalan web server tersebut disebabkan oleh traffic yang berjalan ke satu node yang aktif lalu node tersebut mengalami crash atau down, maka diperlukan jeda agar koneksi dapat terhubung lagi ke web server dengan node yang masih tersedia.

5. SARAN

Dalam penelitian ini tentu masih banyak kekurangan yang perlu dibenahi, seperti:

1. Diperlukannya suatu tools yang dapat memonitoring dan memvisualisasikan log secara realtime agar dapat memberikan informasi dengan mudah.
2. Analisis yang lebih mendalam terjadi bagaimana proses kerja dari scheduler dan mekanisme failover tersebut.

Sehingga sangat diharapkan penelitian selanjutnya bisa mengembangkan analisis failover ini agar lebih detail dan mendalam sehingga dapat dipahami secara utuh dan lengkap.

DAFTAR PUSTAKA

[1] Rexa, M., Data, M. and Yahya, W. (2019) ‘Implementasi Load Balancing Server Web Berbasis Docker Swarm Berdasarkan Penggunaan Sumber Daya Memory Host’, *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer (J-PTIIK) Universitas Brawijaya*, 3(4), pp. 3478–3487.

[2] Bik, F.R. and Asmunin (2017) 'Implementasi Docker Untuk Pengelolaan Banyak Aplikasi Web (Studi Kasus : Jurusan Teknik Informatika UNESA)', *Jurnal Manajemen Informatika*, 7(2), pp. 46–50.

[3] Riko Rudiyanto (2023) 'Implementasi Load Balancing Dan Failover Dengan Border Gateway Protocol Menggunakan Router Juniper (Studi Kasus Stasiun Jombang)', *Program Studi Informatika Program Sarjana Fakulas Teknologi Informasi Universitas Teknologi Digital Indonesia Yogyakarta*, pp. 5–24. Available at: https://eprints.utdi.ac.id/9832/3/3_155410081_BAB_II - riko rudiyanto.pdf.

[4] Prakoso, R.D. and Asmunin (2018) 'Implementasi dan Perbandingan Performa Proxmox Dalam Virtualisasi dengan Tiga Virtual Server', *Jurnal Manajemen Informatika*, 8(1), pp. 79–86. Available at: https://ejournal.unesa.ac.id/index.php/jurnal-manajemen-informatika/article/view/22864.

[5] Sumbogo, Y.T., Data, M. and Siregar, R.A. (2018) 'Implementasi Failover Dan Autoscaling Kontainer Web Server Nginx Pada Docker Menggunakan Kubernetes', *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, 2(12), pp. 6849–6854. Available at: http://j-ptiik.ub.ac.id.

[6] Prasetyo, A. (2017) 'Perancangan Virtualisasi Replikasi Database Pada Arsitektur Cloud Computing', *Research Report*, 0(0), pp. 207–210. Available at: http://research-report.umm.ac.id/index.php/research-report/article/view/1213.

[7] Vaucher, S. (2015) 'Comparing Virtual Machines and Linux Containers', *Université de Neuchâtel* [Preprint]. Available at: https://github.com/docker/docker/blob/master/.