ANALISIS PERBANDINGAN PERFORMANSI RESPON WAKTU WEB SERVER DAN FAILOVER ANTARA KUBERNETES DAN DOCKER SWARM

Ishak M
Program Studi Informatika, Sekolah Tinggi Manajemen dan Ilmu Komputer El Rahma Yogyakarta
e-mail: asky_el@myself.com

Abstrak

Failover sebagai mekanisme dalam mengatasi single failure yang 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*

Abstract

Failover as a mechanism for overcoming single failures that is integrated into Kubernetes and Docker Swarm as container-based cluster servers or container orchestration. However, the differences in failover performance and web server response time on Kubernetes and Docker Swarm are not yet known. Based on this problem, an in-depth analysis is needed regarding the comparison of failover performance and web server response time between Kubernetes and Docker Swarm by carrying out various tests and experiments carried out on each of these clusters. Test results using file upload experiments to determine the response time performance of the web server in each cluster, data found that the response time of Kubernetes was 54.5% faster than that of Docker Swarm. Then in the failover test from the node failure side, Docker Swarm has a very significant average failover time. However, Docker Swarm does not have container management like the management pods found in Kubernetes, whereas during failover testing when the file upload process takes place, it shows that both clusters have the same results, namely the website experienced an error when the test was carried out.
Keywords : *container orchestration, Kubernetes, Docker Swarm, failover*

1. PENDAHULUAN

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].

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.

Beberapa penyebab *server failure* atau kegagalan pada *server* adalah putusnya sumber daya listrik, malfungsi perangkat keras, sistem operasi yang *crash* dan kegagalan jaringan. Salah satu langkah yang dapat dilakukan dalam mengatasi kegagalan pada *server* adalah dengan menggunakan teknik *server cluster* dengan mekanisme *failover*.

Server cluster merupakan sebuah gabungan dari beberapa computer *server* yang digunakan oleh sebuah lembaga maupun organisasi untuk mencapai kebutuhan yang diperlukan oleh server untuk melampaui kemampuan sebuah mesin)[2]. Sedangkan *Failover* merupakan sebuah mekanisme pergantian sebuah layanan seperti *software* atau *hardware* dari sebuah *server* yang sedang mengalami *failure* ke server lain yang tersedia di dalam *cluster*[3].

Failover pada umumnya diimplementasikan untuk tujuan meningkatkan ketersediaan layanan yang disediakan[4]. Elemen klaster akan bekerja dengan node-node redundan yang kemudian digunakan untuk menyediakan layanan saat salah satu elemen klaster mengalami kegagalan[5]. 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*)[6].

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[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

Pada pembuatan rancangan penelitian, metode yang digunakan adalah pengumpulan data dan perancangan sistem. Adapun alat dan bahan adalah sebagai berikut:

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

c. Metode Pengumpulan Data

Metode pengumpulan data yang dapat digunakan adalah:

1) 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.

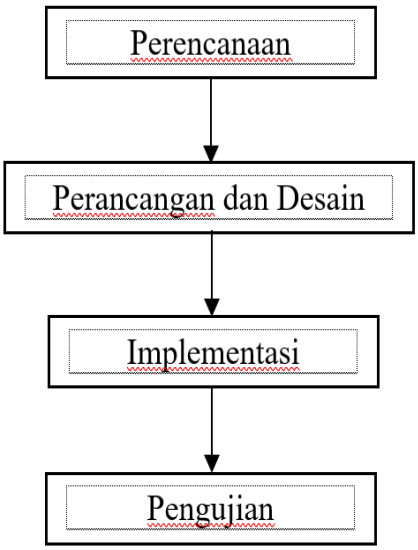
2) Studi Literatur

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.

d. Langkah-Langkah Penelitian

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



Gambar 1. Tahapan Penelitian

1. 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.

2. Perancangan dan Desain

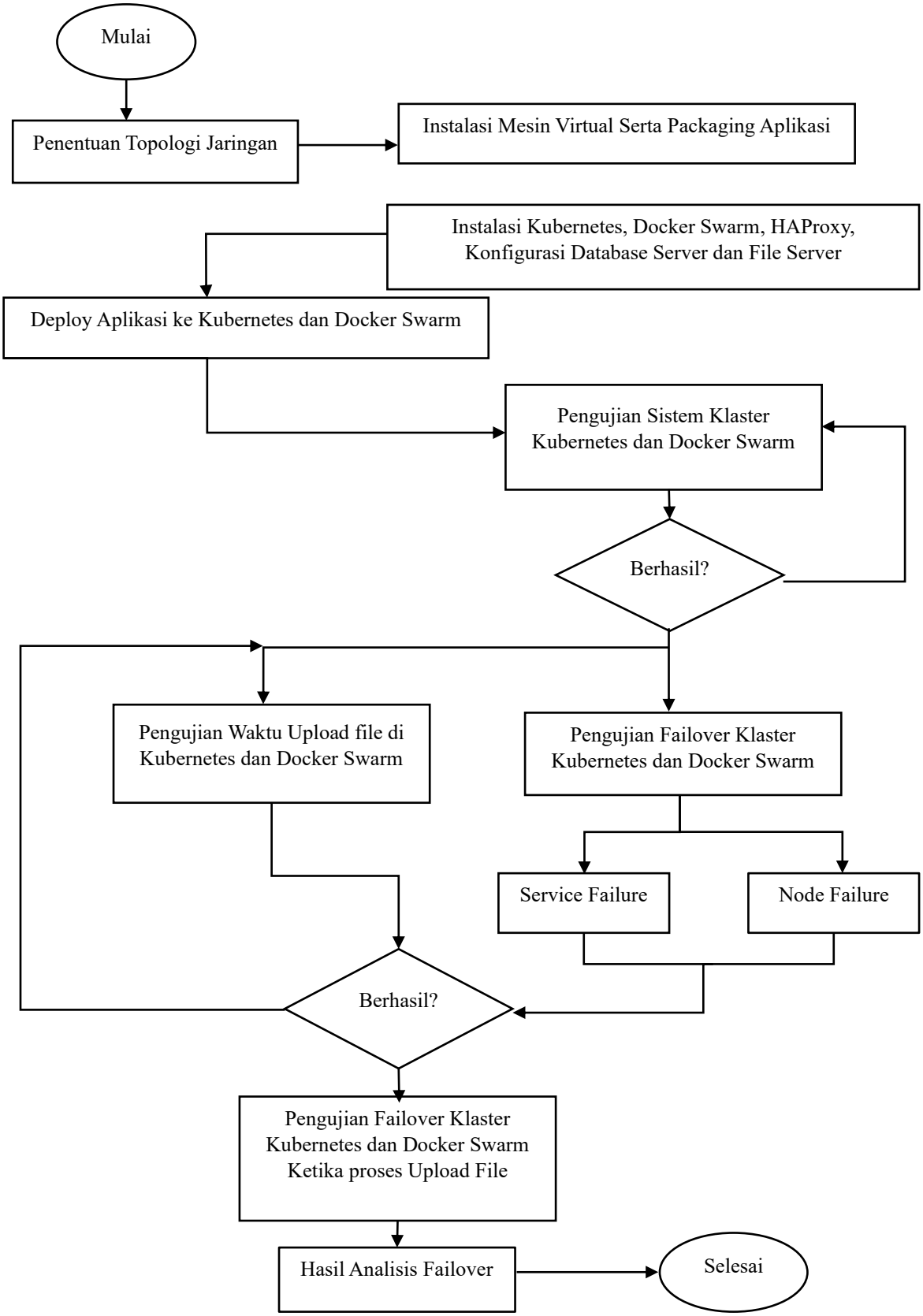
Setelah melakukan pelaksanaan penelitian dan kajian literatur sehingga didapatkan data digital, maka selanjutnya dilakukan perancangan sistem yaitu persiapan mesin, desain topologi jaringan dari server cluster dan perancangan mekanisme yang digunakan.

3. Implementasi

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

4. 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 didalamnya.



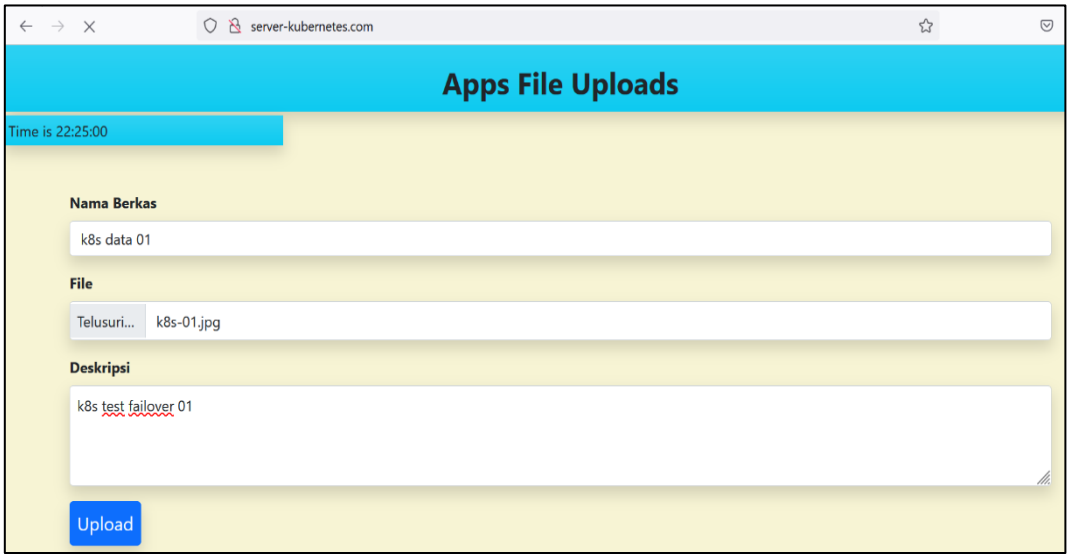
Gambar 2. Alur Penelitian

Alur penelitian seperti yang terdapat pada Gambar 2 merupakan proses keseluruhan rancangan sistem hingga hasil penelitian.

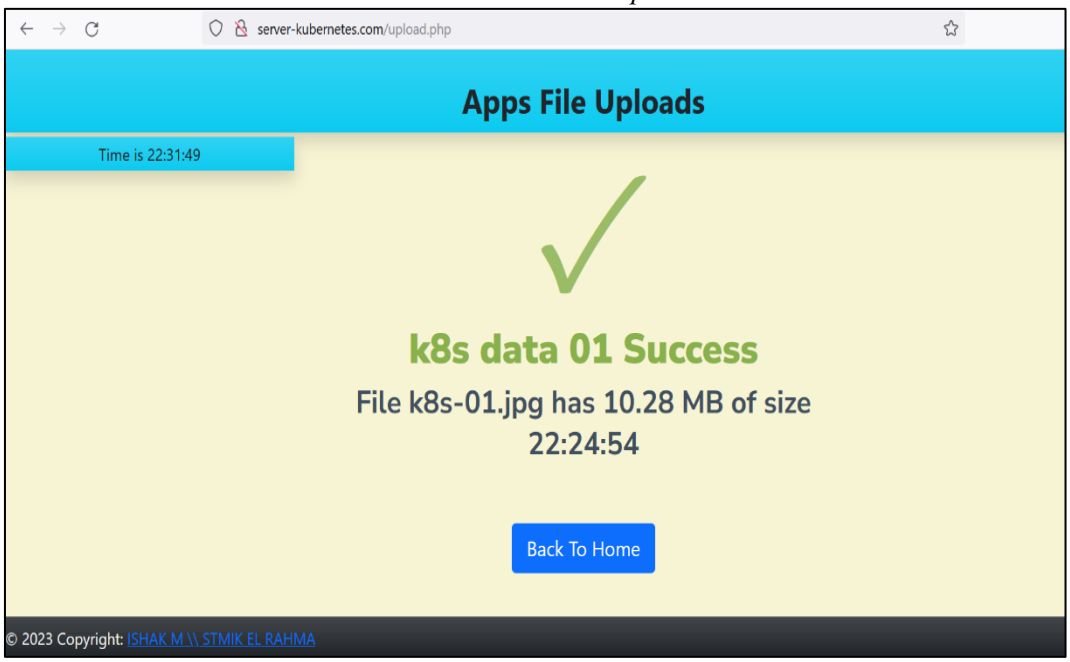
3. HASIL PEMBAHASAN

3.1 Pengujian Upload File Pada Cluster Kubernetes

Pengujian atau percobaan ini dilakukan sebanyak tiga puluh kali dengan menggunakan *file* berukuran 10MB untuk mendapatkan perhitungan durasi yang dihasilkan.



Gambar 3. Proses Awal *Upload File*



Gambar 4. Halaman Selesai *Upload File*

Jika proses upload selesai sesuai dengan Gambar yang tercantum di atas, maka tahap selanjutnya adalah perhitungan waktu selesai dan waktu mulai *upload file* tersebut, hingga ditemukan durasi pada saat proses *upload* berlangsung.

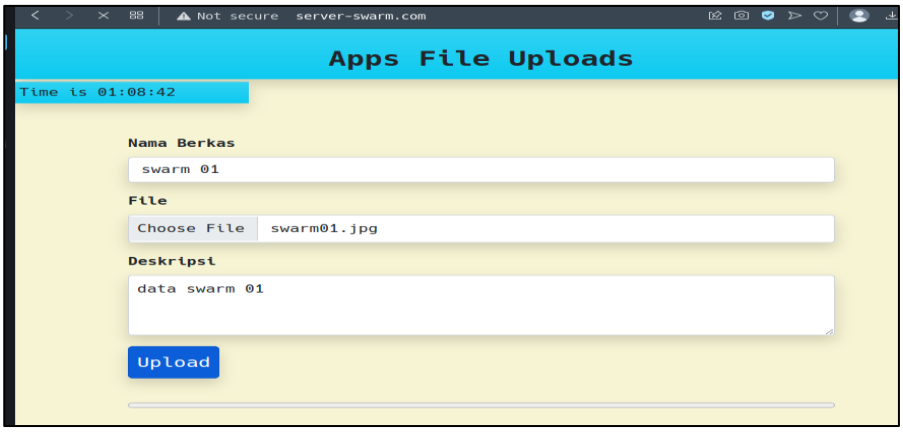
Tabel 5.1 merupakan total durasi yang tercatat dari hasil pengujian yang dilakukan selama tiga puluh kali.

Tabel 1. *Upload File* Pada Kubernetes Cluster

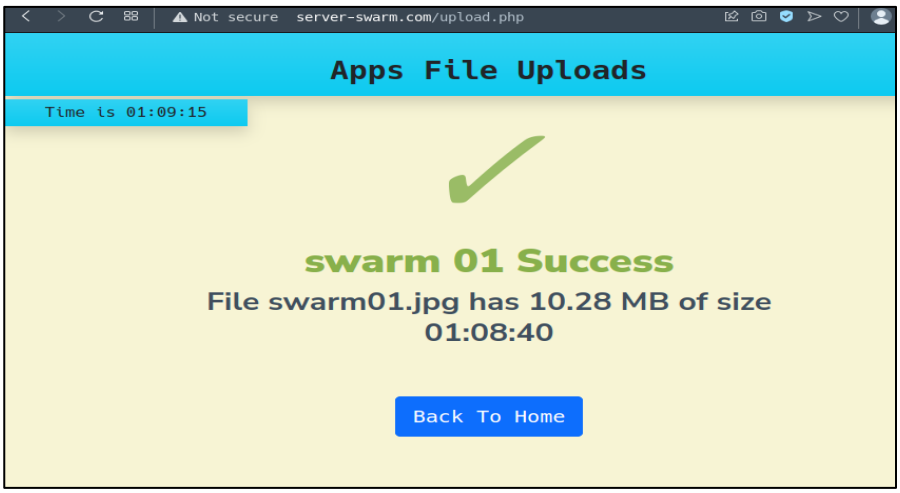
Data Upload	Mulai Upload	Selesai Upload	Lama Upload(detik)
1	00:17:51	00:18:03	00:00:12
2	00:19:31	00:19:41	00:00:10
3	00:20:16	00:20:26	00:00:10
4	00:21:27	00:21:38	00:00:11
5	00:22:38	00:22:51	00:00:13
6	00:23:41	00:23:52	00:00:11
7	00:24:22	00:24:33	00:00:11
8	00:25:40	00:25:51	00:00:11
9	00:26:37	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**

3.2 Pengujian Upload File Pada Docker Swarm

Serupa dengan langkah-langkah yang dilakukan pada kubernetes cluster, pada Docker Swarm cluster juga dilakukan *upload file* berukuran 10MB sebanyak tiga puluh kali.



Gambar 5. Halaman Awal *Upload File*



Gambar 6. Halaman Selesai *Upload File*

Jika prosedur *upload file* telah selesai, maka hasil dari pengujian terlihat pada Tabel 2.

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

Lanjutan Tabel 2.

Data Upload	Waktu Mulai	Waktu Selesai	Lama Upload(detik)
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**

3.3 Analisis Hasil Failover

Pada percobaan yang telah dilakukan menunjukkan bahwa proses *failover* pada *web server* berhasil dilakukan saat sistem mengalami *node failure* ataupun *service failure*. Namun, terdapat perbedaan yang cukup signifikan pada Kubernetes dan Docker Swarm di sisi *service failure*. Docker Swarm diharuskan melakukan *scaling* pada servis setidaknya sebanyak 2 atau lebih servis. Hal ini disebabkan karena Docker Swarm tidak memiliki fitur manajemen *pods* atau *container* seperti yang terdapat pada kubernetes melainkan hanya terdapat *manajemen service*.

Tabel 3. Skenario Analisis Hasil *Failover Kubernetes*

No	Skenario	Hasil	Keterangan
1	Akses *web server* dalam kondisi seluruh *node worker* aktif	Sistem mampu melayani *request*	Sistem *server* berjalan dengan baik
2	Akses *web server* dalam kondisi salah satu *node worker* mengalami kegagalan	Berhasil diakses walaupun salah satu *node worker* mengalami kegagalan/*down*	*pod* pada *node worker* yang *down* berpindah pada *node worker* yang tersedia/aktif.
3	Akses *web server* dalam keadaan seluruh *pod down*	berhasil diakses walaupun seluruh *pod down*	Kubernetes melakukan *service failover* dengan pembuatan *pod* baru pada semua *node worker* aktif

Tabel 3 menunjukan bahwa sistem mampu melayani permintaan dari klien dalam kondisi seluruh *node worker* aktif maupun dalam kondisi salah satu *node worker* tidak aktif dan ketika seluruh *pod down*. *Scheduler* Kubernetes akan menjadwalkan untuk membuat ulang *Pods* yang down atau mengalami crash selama node didalam cluster tersedia. Namun hal tersebut akan berbeda kasus Ketika semua node worker mengalami down.

Tabel 4. Skenario Analisis Hasil *Failover Docker Swarm*

No	Skenario	Hasil	Keterangan
1	Akses *web server* dalam kondisi seluruh *node* aktif	Sistem mampu melayani *request*	Sistem *server* berjalan dengan baik
2	Akses *web server* dalam kondisi salah satu *node worker* mengalami kegagalan/*down*	Berhasil diakses walaupun salah satu *node worker* mengalami kegagalan/*down*	*service* pada *node worker* yang *down* berpindah pada *node worker* yang tersedia/aktif.
3	Akses *web server* dalam kondisi seluruh *node worker* mengalami kegagalan/*down*	Website tidak berhasil	Walaupun node manager mengambil alih traffic service tapi pada LB tidak menemukan node aktif

Tabel 4. menunjukan bahwa sistem mampu melayani permintaan dari klien dalam kondisi seluruh *node worker* aktif maupun dalam kondisi salah satu *node worker* tidak aktif dan ketika *node manager down*.

Hal tersebut dapat tercapai sebagai dampak dari fungsi dari *scaling* yang diberikan oleh Docker Swarm. Jika salah satu *node down* maka *service* akan berganti ke *node* yang sedang aktif.

3.4 Analisis Hasil Pengujian Waktu *Failover*

Berikut adalah data yang didapatkan dari hasil pengujian *failover.* Tabel 5.7 menunjukan hasil rata-rata waktu pengujian ketika *node* diberhentikan secara paksa dan Tabel 5.8 menunjukan hasil rata-rata waktu ketika layanan *web server* diberhentikan secara paksa.

Tabel 5. Hasil Pengujian *Failover Node Failure*

Pengujian	Waktu failover(detik)
1	345
2	344
3	346
4	346
5	345
Rata-rata	**345,2**

Tabel 6. Hasil Pengujian *Failover Service Failure*

Pengujian	Waktu failover(detik)
1	5
2	5
3	7
4	6
5	5
Rata-rata	**5,6**

Tabel 5 dan Tabel 6 menampilkan waktu yang dibutuhkan untuk melakukan *failover* pada sebuah layanan *web server*. Rata-rata waktu yang dibutuhkan adalah 345.02 *second*, atau 5 menit

45 detik untuk *node failure* sedangkan proses *failover* pada pengujian *service failure* memiliki rata-rata waktu yang sangat singkat yaitu 5.6 detik.

Hal ini terjadi karena ketika *node* diberhentikan secara paksa atau mengalami masalah yang menyebabkan *down*, modul Kubelet pada *node worker* tidak dapat berkomunikasi dengan *node master,* sehingga *node master* membutuhkan waktu lebih lama untuk melakukan pemeriksaan.

Tabel 7. Hasil Pengujian *Failover Node Failure Docker Swarm*

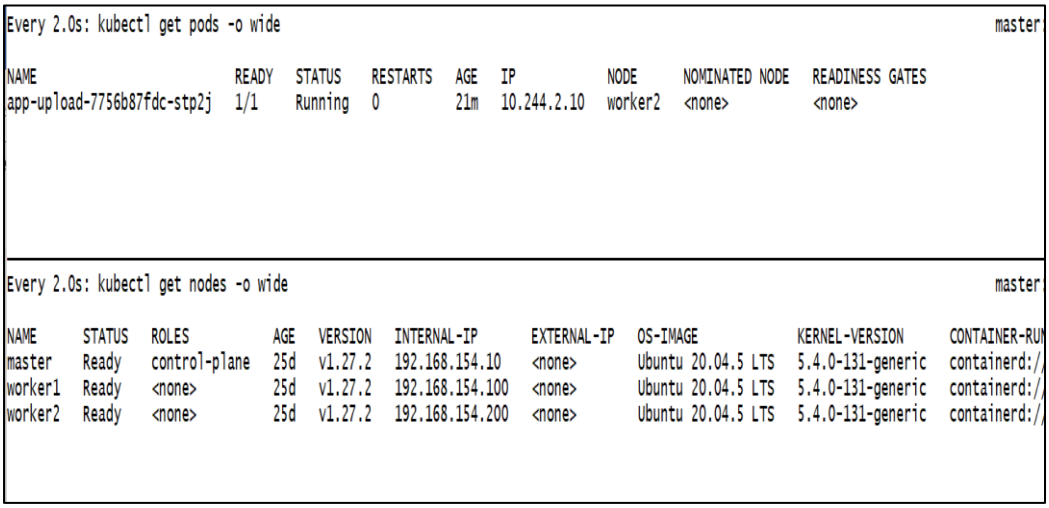
Pengujian	Waktu *Failover* (s)
1	20
2	19
3	21
4	22
5	23
Rata-rata	**21**

Tabel 7 menampilkan keterangan waktu yang dibutuhkan untuk melakukan *failover* pada sebuah layanan *web server*, dengan rata-rata waktu yang dibutuhkan adalah 21 *second.*

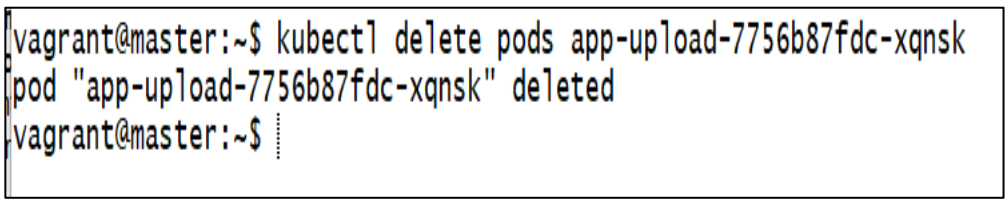
Perbedaan mendasar dan signifikan yang berpengaruh di antara Kubernetes dan Docker Swarm terjadi dikarenakan Docker Swarm memiliki arsitektur dengan orkestrasi yang lebih sederhana dan juga terintegrasi dengan *docker engine* sedangkan Kubernetes mempunyai arsitektur yang lebih kompleks dengan banyak komponen dan lapisan yang akan bekerja ketika *failover* terjadi dan juga *docker engine* di dalam cluster kubernetes merupakan *platform* pihak ketiga.

3.5 Pengujian Failover Ketika Upload di Cluster Kubernetes

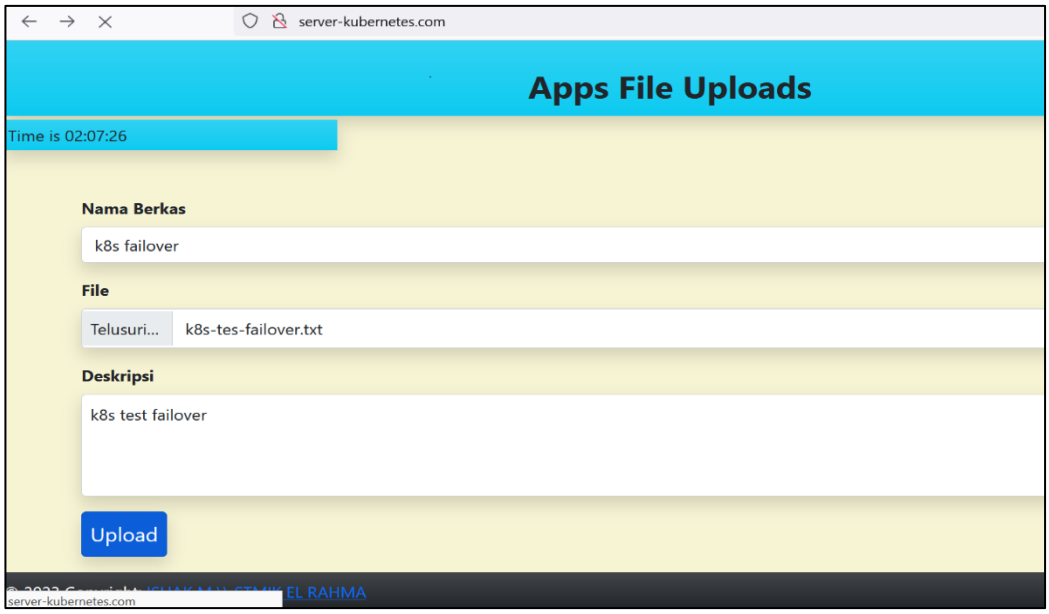
Pada pengujian *failover* ketika *upload file* ini, akan menitik beratkan pada *monitoring loadbalancer* dan *tool watch* di *server* pada terminal. Ketika proses *upload file* sedang berlangsung, terlebih dahulu akan dilakukan pemantauan pada *loadbalancer* dan *tool watch* untuk melihat perbedaan ketika melakukan aksi penghapusan *pod* dan memberhentikan *node worker* yang sedang berjalan.



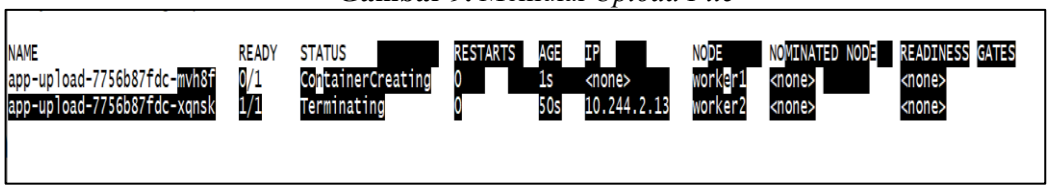
Gambar 7. Status Awal *Monitoring Tool Watch*



Gambar 8. *Kubectl Delete Pods App-Upload*



Gambar 9. Memulai *Upload File*



Gambar 10. Proses Pembuatan *Pod* Baru

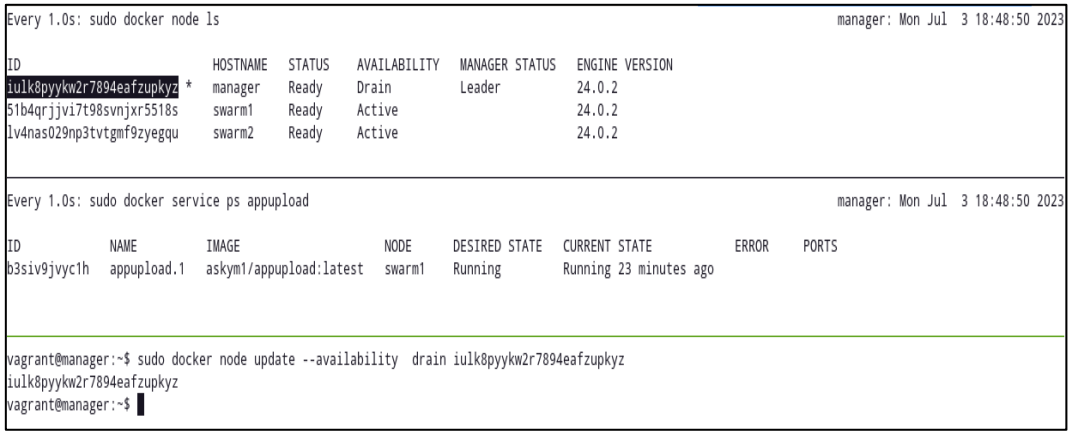
Gambar 11. Tampilan *Website* Setelah Penghapusan *Pod*

Website mengalami error selama proses upload file berlangsung dan mengalami failover.

3.6 Pengujian Failover Ketika Upload di Cluster Docker Swarm

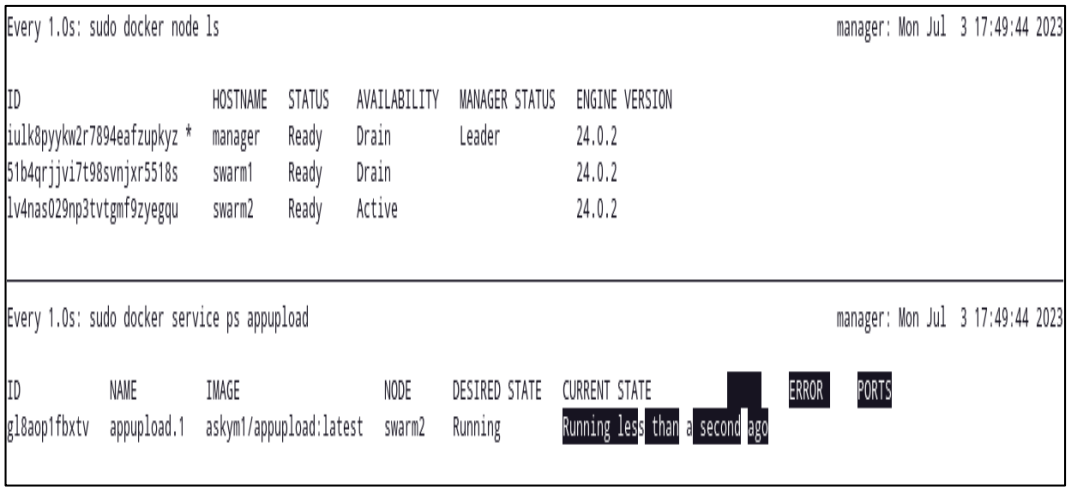
a. Mode *Non-Scaling*

Secara *default,* ketika pertama kali membuat *service,* Docker Swarm akan menjalankan servis pada *node manager*, sehingga perlu untuk menjalankan *drain* dengan perintah *sudo docker node update –availability drain node_id* agar service berjalan pada *node worker.*

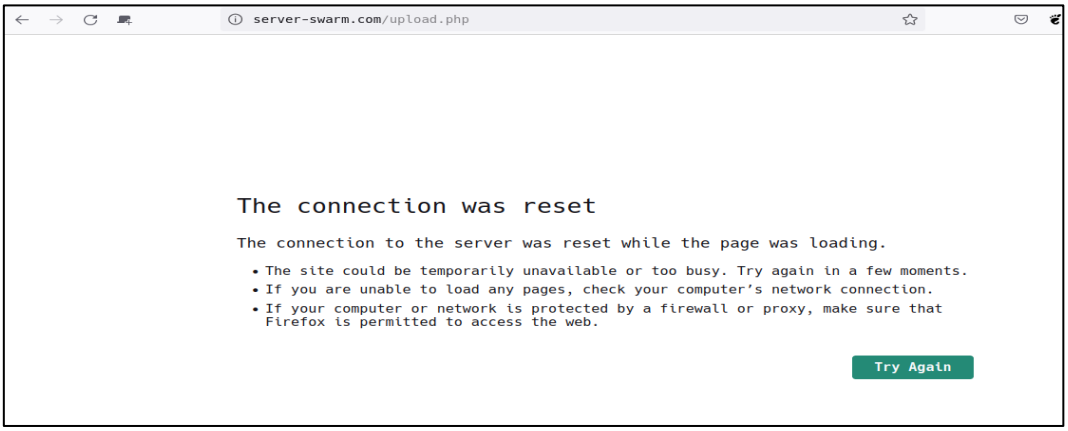


Gambar 12. *Drain Node Manager*

Selanjutnya masuk ke website server-swarm.com lalu beralih ke proses *upload file*. Ketika proses *upload* sedang berjalan, jalankan perintah *sudo docker node update –availability drain node_id* agar *node swarm1* menjadi tidak tersedia dalam penanganan *service* sehingga membuat *service* beralih ke swarm2.



Gambar 13. Status Setelah *Drain*

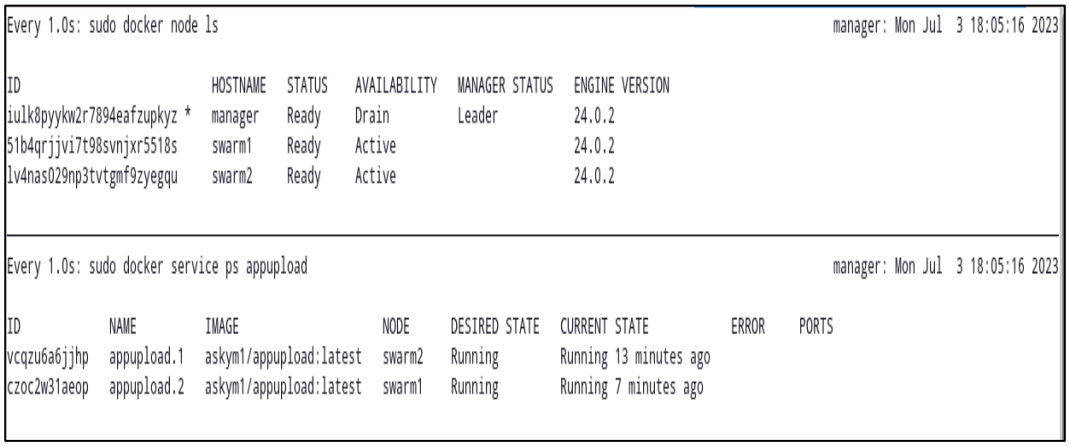


Gambar 14. *Website* Setelah *Failover*

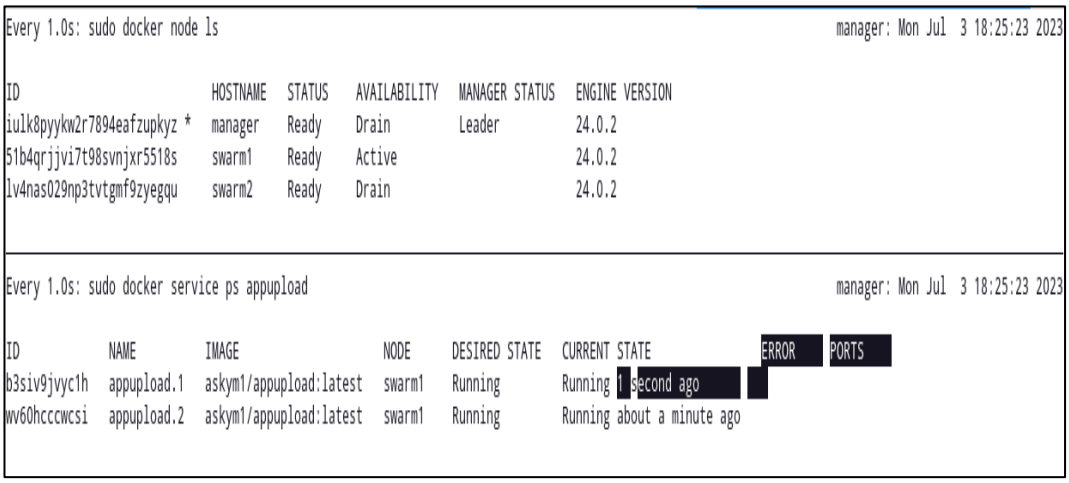
Website mengalami error selama upload file berlangsung dan mengalami failover.

b. **Mode *Horizontal Scaling***

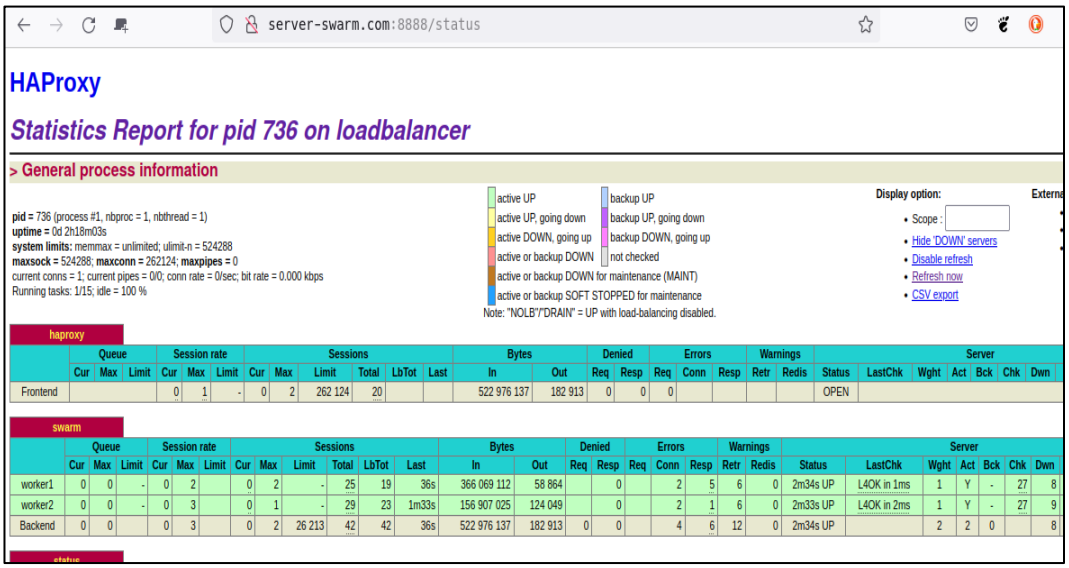
Pada mode ini, langkah pertama yang dilakukan adalah menjalankan perintah *sudo docker service scale appupload=2.* Angka 2 menyesuaikan dengan semua *node worker* yang tersedia di *cluster* maupun di *loadbalancer*.



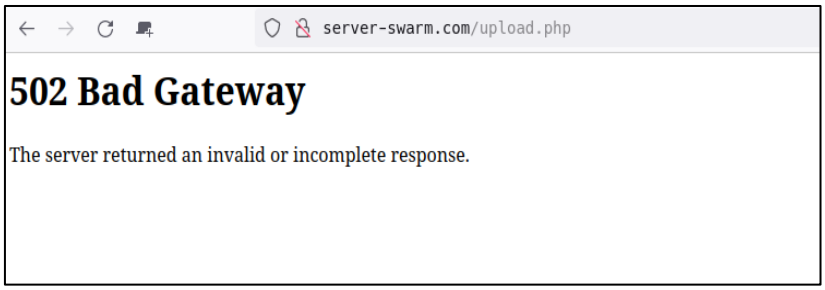
Gambar 15. Status Awal *Node* dan *Service*



Gambar 16. Status *Node* dan *Service* Setelah *Drain Worker Swarm2*



Gambar 17. *Loadbalancer* Setelah *Drain Worker Swarm2*



Gambar 18. Status *Website* Setelah *Failover*

Website mengalami error selama proses upload file berlangsung dan mengalami failover. Lalu pada *dashboard monitoring loadbalancer*, *failover* yang terjadi tidak akan memengaruhi *loadbalancer* dikarenakan *service* telah diambil alih oleh *node* yang masih aktif tetapi pada *website*, koneksi tetap terputus ke *node worker* sehingga menyebabkan *website down*.

4. KESIMPULAN

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%. Sedangkan 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.

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.

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

Diperlukannya suatu tools yang dapat memonitoring dan memvisualisasikan log secara realtime agar dapat memberikan informasi yang sederhana dan mudah dimengerti dan juga analisis yang lebih mendalam terkait 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] M. Rexa, M. Data, and W. Yahya, “Implementasi Load Balancing Server Web Berbasis Docker Swarm Berdasarkan Penggunaan Sumber Daya Memory Host,” *J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya*, vol. 3, no. 4, pp. 3478–3487, 2019.

[2] A. Frederius, M. Data, and W. Yahya, “Implementasi Penyimpanan Data Persisten pada Docker Swarm Menggunakan Network File System (NFS),” *J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya*, vol. 3, no. 2, pp. 9088–9096, 2019.

[3] Y. T. Sumbogo, M. Data, and R. A. Siregar, “Implementasi Failover Dan Autoscaling Kontainer Web Server Nginx Pada Docker Menggunakan Kubernetes,” *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 2, no. 12, pp. 6849–6854, 2018, [Online]. Available: http://j-ptiik.ub.ac.id

[4] Y. B. Ginting *et al.*, “Implementasi Metode Failover Sebagai Backup Server Pada Arsitektur Load Balancer,” *Coding J. Komput. dan Apl.*, vol. 09, no. 02, pp. 198–210, 2021.

[5] M. I. Wahyuddin and M. Jejen, “Perancangan dan Implementasi High Availability Cluster Web Server Berbasis DBRD dan Heartbeat,” pp. 205–211, 2016, doi: 10.5614/sniko.2015.30.

[6] S. D. - AMIK BSI Pontianak and Y. - AMIK BSI Jakarta, “Penerapan Jaringan Lan Dengan Sistem Redudancy Static Route Menggunakan Router Mikrotik Pada Pt. Sistem Aksesindo Perdana Jakarta,” *Evolusi J. Sains dan Manaj.*, vol. 6, no. 1, pp. 114–119, 2018, doi: 10.31294/evolusi.v6i1.3589.

[7] S. E. Prasetyo and Y. Salimin, “Analisis Perbandingan Performa Web Server Docker Swarm dengan Kubernetes Cluster,” *Comb. - Conf. Manag. Business, Innov. Educ. Soc. Sci.*, vol. 1, no. 1, pp. 825–833, 2021, [Online]. Available: https://journal.uib.ac.id/index.php/combines/article/view/4512