Analisis Perbandingan High Availibility Pada Web Server Menggunakan Orchestration Tool Kubernetes Dan Docker Swarm

Wahyu Aldiwidianto[1], I Gusti Lanang Eka Prismana[2]

[1,2] Jurusan Teknik Informatika, Fakultas Teknik, Universitas Negeri Surabaya
[1]wahyu.18043@mhs.unesa.ac.id
[2]lanangprismana@unesa.ac.id

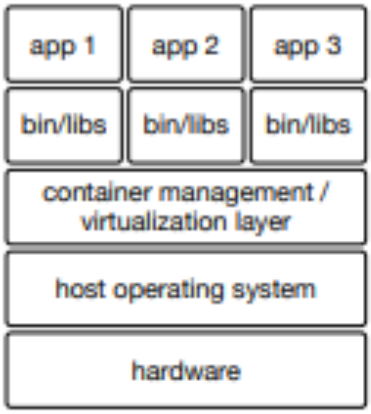
***Abstrak*— Penelitian ini bertujuan untuk mengetahui penggunaan sumber daya CPU serta efektivitas orchestration tool Docker Swarm dan Microk8s dalam menangani downtime menggunakan teknik failover untuk mencapai high availability, keduanya menjalankan service web server didalam container Docker. Penelitian ini berjenis expremintal design yang mencoba menerapkan sistem high availibility pada web server menggunakan orchestration tool Microk8s dan Docker Swarm. Rancangan high availability server diimplementasikan dengan 3 virtual server yang dijalankan diatas sistem operasi ubuntu. Failover bekerja jika server master mati dan secara otomatis akan digantikan dengan server backup yang telah dikonfigurasi pada perangkat lunak Keepalived. Skema pengujian dilakukan dengan mengirim HTTP request ke server menggunakan perangkat lunak Apache Jmeter, sample request yang dikirim berjumlah 30000 dengan waktu 10 menit, salah satu server akan dimatikan untuk menguji seberapa lama downtime dan penggunaan CPU pada server yang masih aktif. Hasil penelitian menunjukan Docker Swarm lebih baik dalam menangani downtime dengan hasil rata-rata waktu henti mencapai 6,4 detik, sedangkan Microk8s rata-rata downtime 43,7 detik. Dalam penggunaan CPU loadbalancer berjalan dengan cukup baik pada kedua orchestration tool pada saat downtime terjadi. Namun efektivitas penggunaan CPU jauh lebih baik pada Docker Swarm dengan hasil rata-rata CPU usage sebesar 18,202% pada server 1 dan 13,482% pada server 2, sedangkan Micok8s rata-rata penggunaan CPU pada server 1 sebesar 39,2% dan server 2 32,689%. Maka dapat disimpulkan kedua orchestration dapat mengatasi downtime dengan cukup stabil dari keseluruhan pengujian, namun terdapat perbedaan hasil downtime yang cukup signifikan yang dipengaruhi salah satunya oleh penggunaan CPU yang kurang efisien.**

***Kata Kunci*— High Availability, Microk8s, Docker Swarm, Failover, Web Server**

I. PENDAHULUAN

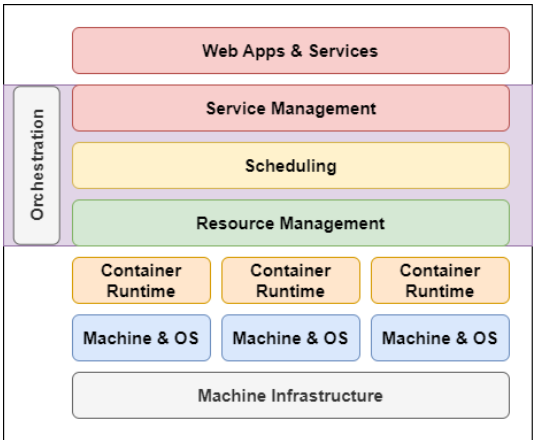
. Di era digital ini, terjadi peningkatan kebutuhan aplikasi berbasis *web* yang sejalan dengan meningkatnya pengguna internet di seluruh dunia [1]. Masalah *downtime* bisa terjadi jika *server* mengalami kegagalan pada sisi perangkat keras maupun perangkat lunak yang menyebabkan *website* tidak dapat diakses oleh pengguna. *Downtime* adalah waktu dimana sistem tidak dapat digunakan untuk menjalankan fungsinya sesuai yang diharapkan. *Downtime* dapat terjadi karena kerusakan komponen perangkat keras atau perangkat lunak, adanya jaringan yang putus, *traffic* yang terlalu tinggi, dan juga serangan peretas [2].

Untuk dapat menyelesaikan permasalahan *downtime* maka diperlukan sebuah sistem yaitu *high availablity*. *High Availability* atau Ketersediaan Tinggi adalah sistem yang dapat beroperasi terus menerus tanpa kegagalan dengan tingkat kinerja dan kualitas operasional yang tinggi [3]. *Failover* dapat digunakan untuk mengatasi *downtime*, dengan menyediakan *server* cadangan yang dapat menggantikan server utama ketika mengalami *downtime*. *Failover* bekerja dengan mendeteksi *server* mana yang sedang mengalami *downtime*, secara otomatis *server* yang mengalami downtime digantikan dengan dengan *server* aktif lain dalam satu cluster yang sama [3].



Gbr 1. Arsitektur *Container*

Salah satu teknologi untuk mengimplementasikan server adalah *container*. *Container* merupakan sistem yang memungkinkan Anda untuk membuat layanan dan sumber daya yang berbeda dengan membaginya dalam sistem operasi dan menjalankannya di *libraries*, *drivers*, dan *binaries* yang berbeda [4]. *Docker* merupakan aplikasi *container engine* yang bersifat *open source*. Dalam penerapannya, *Docker* sangat mudah dan praktis untuk mengembangkan aplikasi menggunakan virtualisasi level OS (*Operating System*) yang tergabung dalam sebuah paket yang disebut *container* [5]. Docker memiliki dukungan terhadap *container orchestration* Docker Swarm dan Kubernetes, yang membantu *developer* dalam memilih *orchestration tool* yang dapat digunakan.



Gbr 2. Arsitektur Container Orchestration

Container orchestration merupakan sebuah sistem *open source* yang bertindak sebagai pengontrol untuk mengelola beberapa *cluster container* dan menyederhanakan pengelolaan *container* [6]. Kedua *container* orchestration tersebut mendukung sistem *high availability*. *Container orchestration* memungkinkan untuk menentukan penyediaan otomatis dan mengubah alur kerja manajemen untuk beroperasi sehingga selalu memberikan kebijakan dan tingkat layanan yang disepakati.

Platform MicroK8s (mK8s) menyediakan distribusi yang kompatibel dengan K8S dengan memodifikasi dan mengatur ulang komponen penting [6]. MicroK8s adalah sistem sumber terbuka untuk mengotomatisasi penerapan, penskalaan, dan pengelolaan aplikasi dalam *container*. Microk8s menyediakan fungsionalitas komponen inti Kubernetes dalam *footprint* kecil dan dapat diskalakan dari satu *node* ke *multi-server* produksi dengan *high availability*.

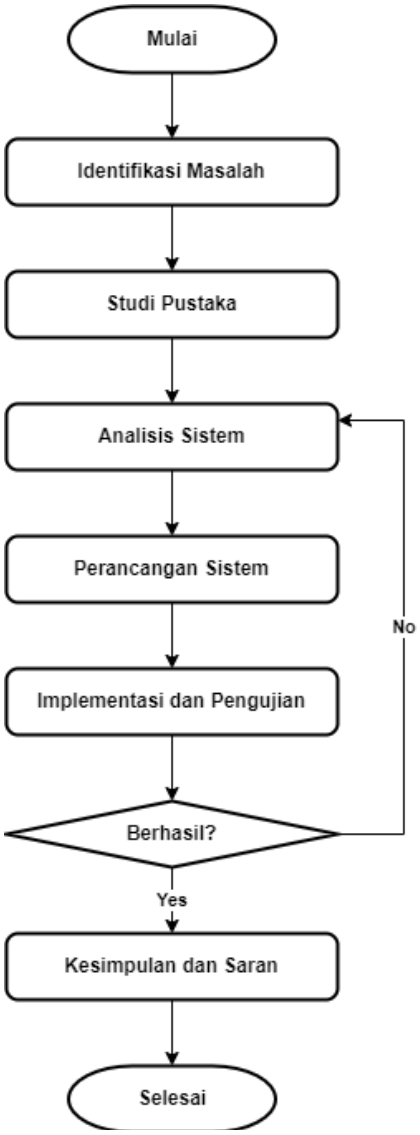
Docker Swarm merupakan alat orkestrasi yang dikembangkan oleh Docker sendiri. Cluster Docker Swarm memungkinkan seseorang untuk menambahkan jumlah node yang tidak terbatas, dan Docker memungkinkan seseorang untuk menjalankan container tanpa batas pada node, hal ini memberikan pengujian skala penuh yang mendekati kondisi nyata [7].

Penelitian ini bertujuan untuk mengetahui *platform* mana yang paling efektif dalam menangani masalah *downtime*. Selain akan diketahui efetifitas penggunaan CPU selama proses pengujian berlangsung. Harapannya penelitian ini dapat digunakan sebagai bahan acuan dalam memilih *platform* orkestrasi yang paling efektif antara Docker Swarm dan Microk8s.

II. METODE PENELITIAN

Penelitian yang dilakukan merupakan penelitian jenis *expremintal design* yang mencoba menerapkan sistem *high availability* menggunakan *orchestration tool* Microk8s dan Docker Swarm yang diterapkan pada *service web server*. Hasil yang didapatkan pada penelitian ini berupa nilai *CPU usage* dan *downtime* yang digunakan sebagai variabel perbandingan antara kedua orchestration tool.

Dalam penelitian ini terdapat beberapa tahap dalam pengerjaan untuk digunakan sebagai awal dalam menyelesaikan penelitian. Tahapan tersebut digambarkan pada diagram berikut :



Gbr 3. Tahapan Penelitian

A. Analisis Kebutuhan Sistem

Pada tahapan ini akan dilakukan analisis sistem yang dapat digunakan sebagai bahan acuan dalam membangun rancangan high availibility. Untuk mempermudah menganalisis sebuah sistem dibutuhkan dua jenis kebutuhan. Kebutuhan fungsional dan kebutuhan nonfungsional yang dijabarkan sebagai berikut :

1) Kebutuhan Fungsional

Kebutuhan fungsional adalah kebutuhan yang berisi proses-proses apa saja yang nantinya dilakukan oleh sistem.

- Docker images Nginx, Mysql, dan PHP dapat berjalan baik di dalam container Docker maupun dalam pod Kuberntes untuk dapat menjalankan environment web service secara maksimal.

- Docker Swarm dan Microk8s akan memanajemen grup dan mengatur replikasi serta high availability pada server.
- Keepalived akan mengatasi mekanisme failover ketika salah satu node server mati.

2) Kebutuhan Nonfungsional

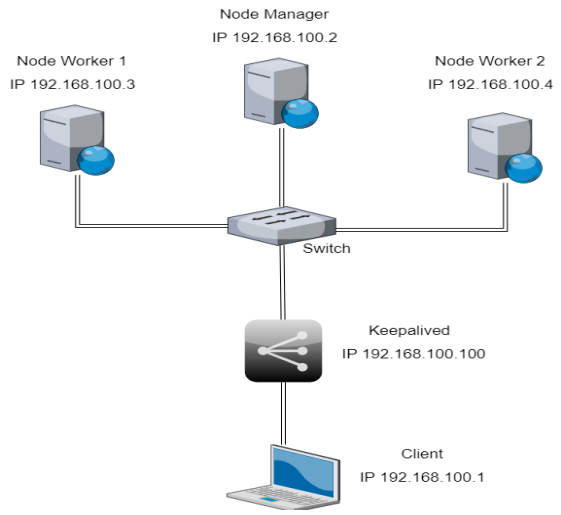
Kebutuhan non-fungsional adalah kebutuhan yang menitikberatkan pada properti yang dimiliki oleh sistem.

a) Spesifikasi Komputer Host
- Asus TUF FX505DT
- Processor AMD Ryzen 7 3750H 8 Core
- RAM 12 GB
- SSD 500 GB
- Sistem Operasi Windows 10 64-bit
- Virtual Box
- Virtual Ubuntu
- Apache Jmeter

b) Spesifikasi Virtual Server
- Processor 1 Core
- RAM 1024 MB
- Penyimpanan 20 GB
- Sistem Operasi Ubuntu 22.04
- Docker Swarm atau Microk8s
- Keepalived
- Systat

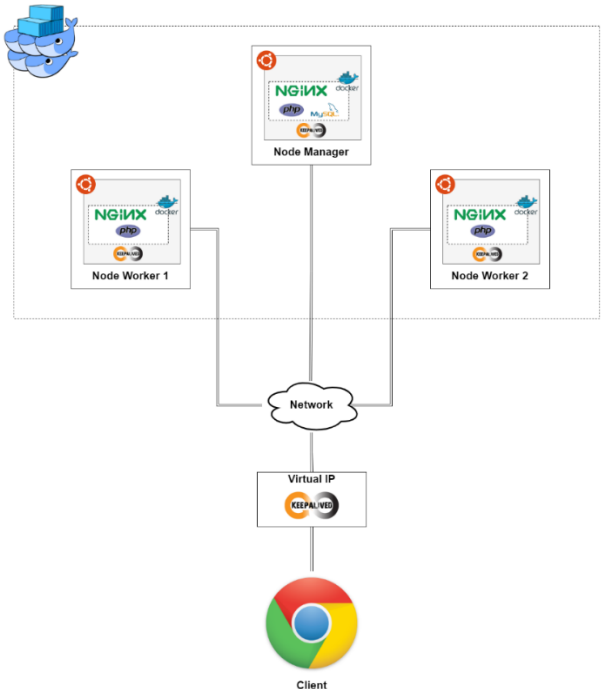
B. Rancangan Topologi Jaringan

.



Gbr 4. Topologi Jaringan Komputer

Pada tahap ini akan digambarkan topologi jaringan yang dibutuhkan untuk membangun *service web server* menggunakan *container* dengan tingkat ketersediaan yang tinggi. Untuk dapat membuat sebuah *multi-server node manager* dan *worker* harus bisa terhubung dalam satu jaringan. Pada kasus ini semua *node* memiliki konfigurasi IP dengan kelas yang sama. *Node manager* dapat mengundang 2 *node worker* untuk masuk dalam satu *multi-server*. Virtual IP menggunakan Keepalived akan menjadi jembatan antara *client* dan *server* untuk bisa saling terhubung. Konfigurasi yang diberikan di node server berdasarkan kebutuhan dan rancangan spesifikasi yang dibuat.

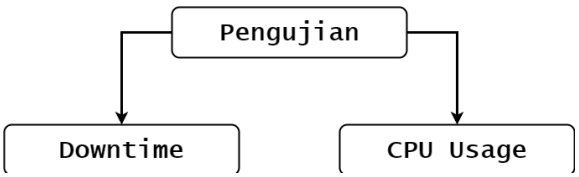


Gbr 5. Mekanisme Sistem High Availabilty Container

Lingkungan *multi-server* yang dibangun baik menggunakan Docker Swarm maupun Microk8s akan memiliki pola konfigurasi yang hampir sama. Keduanya akan dijalankan pada sistem operasi Linux Ubuntu *Server* versi 22.04 LTS. Untuk dapat menangani *failover,* Keepalived akan diinstall pada setiap *node*. Docker menjalankan *container* Nginx, PHP, MYSQL dan dimanajemen dnegan *orchestration tool* Docker Swarm dan Microk8s

Pada penelitian ini *container* Nginx dan PHP dikembangkan dalam satu lingkungan *image* Docker oleh Trafex. *Image* yang telah dibangun oleh Trafex ini memudahkan dalam pengembangan layanan web yang membutuhkan bahasa pemrograman PHP dalam satu lingkungan *web server* Nginx. Sedangkan image MYSQL diambil dari Docker Official Image. Dengan pemilihan *image* Docker tersebut diharapkan lingkungan *web service* yang akan dibangun dapat berjalan dengan baik dan maksimal.

C. Skema Pengujian



Gbr 6. Skema Pengujian

Model sistem yang selesai dibuat akan di uji CPU *usage* dan *downtime*. Skema pengujian *downtime* dilakukan dengan mematikan *node server* yang menjadi *gateway* untuk *web server*, Keepalived akan menangani proses *failover* dengan memindahkan IP yang ke *node server* cadangan. Tugas *orchestration tool* untuk menangani sikronisasi *web server* agar tetap aktif dan berjalan seperti sebelumnya. Penelitian ini menggunakan perangkat lunak Apache Jmeter untuk membantu mengukur downtime pada server.

Pengujian pada penggunaan CPU setiap node dilakukan dengan load testing menggunakan Apache Jmeter. Untuk dapat mengetahui hasil penggunaan CPU maka digunakan perangkat lunak Systat. Pengujian CPU dilakukan bersamaan dengan pengujian downtime, dimana *server backup* yang aktif akan dimonitor rata-rata penggunaan CPU selama waktu yang ditentukan.

Pengujian dilakukan dengan rentang waktu 10 menit dan dilakukan sebanyak 10 kali dengan harapan data yang diambil dapat tervalidasi. Seluruh pengujian yang dilakukan guna untuk mencari perbandingan sistem mana yang paling baik dalam menangani proses *failover*, baik dari segi sistem maupun penggunaan sumber daya perangkat keras komputer *server*. Yang mana tujuan pengujian ini agar didapatkan hasil perbandingan paling baik dari kedua *orchestration tool* yang digunakan pada peneilitian ini..

III. Hasil Dan Pembahasan

Setelah melakukan perancangan sistem akan dijelasnkan mengenai hasil implementasi dan pengujian *high availibilty* pada Docker Swarm dan Microk8s. Adapun rangkaian proses penelitian dimulai dari implementasi hingga pengujian sistem akan dijelaskan sebagai berikut.

A. *Implementasi Sistem*

1) Konfigurasi Keepalived

Keepalived merupakan aplikasi dengan tingkat kehandalan tinggi yang khusus dikembangkan dalam sistem Linux untuk mewujudkan *Virtual Router Redundancy Protocol* [9] Keepalived terdiri dari satu *server* utama dan beberapa *server* cadangan, dan konfigurasi layanan yang sama digunakan pada *server* utama dan *server* cadangan, menyediakan layanan dengan menggunakan satu alamat IP virtual yang secara otomatis berpindah ke *server* cadangan ketika *server* utama mengalami kegagalan[9].

Tabel 1. Perintah Mengedit Konfigurasi Keepalived

root@node1
sudo pico /etc/keepalived/keepalived.conf

Perintah pada tabel 1 dikonfigurasi pada setiap node yang tergabung dalam *multi-server*. Konfigurasi IP yang digunakan pada setiap *node* telah diatur sesuai dengan rancangan topologi jaringan. *Node* 2 berperan sebagai *Master*, dengan IP 192.168.100.100 dan menjadi *gateway* untuk *client* dapat mengakses *website*. *Node* 1 dan 3 berperan sebagai *backup*, pada kasus ini jika *node* 2 mati maka secara otomatis *node* 1 atau 3 akan menggantikannya. Hal ini bisa terjadi karena nilai *priority* pada konfigurasi diatur lebih besar pada *node* 2, semakin besar nilai *priority* maka akan lebih diprioritaskan. Konfigurasi Keepalived dapat dilakukan dengan mengedit *file* keepalived.conf.

```
vrrp_instance VI_1 {
  state BACKUP
  interface enp0s8
  virtual_router_id 51
  priority 253
  advert_int 1
  authentication {
    auth_type PASS
    auth_pass 12345
  }
  virtual_ipaddress {
    192.168.100.100/24
  }
}
```

```
vrrp_instance VI_1 {
  state MASTER
  interface enp0s8
  virtual_router_id 51
  priority 255
  advert_int 1
  authentication {
    auth_type PASS
    auth_pass 12345
  }
  virtual_ipaddress {
    192.168.100.100/24
  }
}
```

```
vrrp_instance VI_1 {
  state BACKUP
  interface enp0s8
  virtual_router_id 51
  priority 254
  advert_int 1
  authentication {
    auth_type PASS
    auth_pass 12345
  }
  virtual_ipaddress {
    192.168.100.100/24
  }
}
```

Gbr 7. Source Code Konfigurasi Keepalived

Dapat dilihat dari gambar 7 diatas *state* menyediakan metode untuk menyesuaikan prioritas pada sistem *failover*, dimana *node* 1 dan *node* 3 sebagai *backup server* dan *node* 2 sebagai *master server*. *Node* 2 di atur sebagai *master* untuk menjadi *gateway request* pada *web server*. *Interface* yang terhubung kedalam jaringan adalah enp0s8 pada setiap *node server*. *Virtual_router_id* memiliki nilai yang sama karena jaringan dibangun untuk membuat sebuah *multi-server*. *Advert_int* bernilai 1 yang berarti Keepalived mengirimkan sinyal ke seluruh *node* setiap 1 detik, hal ini memicu perubahan *server* dari *backup* menjadi *master* ketika *node* utama tidak merespon. *Authentication* menentukan jenis autentikasi id (*auth_type*) dan kata sandi (*auth_pass*) yang digunakan untuk mengautentikasi *server* yang nantinya menjadi media sinkronisasi *failover*. *Virtual_ipaddress* berisi IP 192.168.100.100/24, IP ini dipilih karena tidak digunakan dalam jaringan.

2) Konfigurasi Docker Swarm

Untuk dapat menjalankan Docker Swarm hanya perlu menginstall Docker *Engine* pada Ubuntu. Docker Swarm di bangun dalam satu lingkungan Docker, jadi ketika melakukan instalasi Docker secara otomatis Docker Swarm juga akan terinstal. Pada penelitian ini digunakan Docker versi 20.10.21.

Setelah *node* server dan Docker siap dijalankan, langkah selanjutnya adalah membuat *multi-server* dengan Docker Swarm. Inisialisasi *multi-server* dilakukan pada *instances manager*.

Tabel 2. Perintah Inisialisasi Node Manager

root@node1
docker swarm init –-advertise-addr 192.168.100.2

Inisialisasi pada *node manager* selesai, multi-server dapat dibuat dengan memasukan perintah pada table 3. Token diproduksi secara otomatis oleh Docker Swarm dan tidak akan berubah hingga node manager meninggalkan multi-server.

Node yang diundang akan secara otomatis menjadi worker pada multi-server.

Tabel 3. Perintah Inisialisasi Node Worker

root@node2, node 3
docker swarm join –token (Token Docker Swarm) 192.168.100.2:2377

Node yang telah bergabung pada *multi-server* Docker Swarm dapat dilihat dengan mengetikan perintah *docker node ls*. Dapat dilihat data setiap *node* yang terhubung dalam *multi-server* seperti ID, *hostname*, *status*, *availability*, *manager status*, dan *engine version*. *Multi-server* Docker Swarm telah selesai dikonfigurasi dan siap untuk digunakan.

```
root@node1:~# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS
olxcoepiuqxnyj6bvszfrvut6 *   node1      Ready     Active         Leader
r5nsi3p4mfdqjawnujqt9t0cp     node2      Ready     Active
o5gpt4yag7cz2gcbtd757tco3     node3      Ready     Active
```

Gbr 8. Daftar Node Dalam Multi-server Docker Swarm

Selesai mengkonfigurasi *Multi-server*, maka yang dilakukan selanjutnya adalah membuat *file* konfigurasi *compose*. *File* dengan format .yml ini nantinya dieksekusi untuk menjalankan service Nginx-PHP dan MYSQL. Untuk menyunting *file* Docker Compose pada Ubuntu menggunakan perintah dapat dilihat pada table 4.

Tabel 4 Perintah Untuk Mengedit Konfigurasi Compose File

root@node1
sudo pico docker-compose.yml

Pada penelitian ini konfigurasi Nginx-PHP dan Mysql dibuat dalam satu *file* yang sama. *Compose file* menggunakan versi 3.8, sedangkan *service web server* diberi nama nginx-php yang menggunakan image trafex/php-nginx. *Service* nginx-php diatur untuk bergantung pada *service db*, dimana *service web server* hanya aktif ketika *service db* juga aktif. Data *web* disimpan di penyimpan lokal yang dapat diakses *web server* dengan *volumes*. *Port default image* adalah 8080 yang di *expose* menjadi *port* 80, hal ini bertujuan agar *web server* dapat diakses secara eksternal. *Web server* di *deploy* secara global dimana setiap *node* memiliki 1 replika *container*.

```
version: '3.8'

services:
  nginx-php:
    image: trafex/php-nginx
    depends_on:
      - db
    volumes:
      - ./web/web-server-tugas-akhir:/var/www/html/
    ports:
      - 80:8080
    deploy:
      mode: global
```

Gbr 9. Source Code Konfigurasi Compose Web Server

Service db menggunakan image Mysql sebagai *database server*. *Database server* ini digunakan untuk mengatahui seberapa efektif penggunaan sumber daya *orchestration tool* pada saat menjalankan *web server* dan *database server* dalam satu layanan. Inisialisasi variabel pada *environment* yang berisikan *root password* dan nama *database*. *Volumes* membuat data yang tersimpan pada Mysql akan disalin pada penyimpanan lokal secara otomatis. *Port* di *expose* dari bawaannya 9906 menjadi 3306.

```
db:
  image: mysql
  environment:
    MYSQL_ROOT_PASSWORD: 123
    MYSQL_DATABASE: waldbag
  volumes:
    - ./sql:/var/lib/mysql
  ports:
    - 9906:3306
```

Gbr 10. Source Code Konfigurasi Compose Database Server

Compose file yang telah selesai dibuat dapat dieksekusi dengan menggunakan perintah yang dapat dilihat pada tabel 5. Perintah ini akan menyebarkan *service* yang telah dibuat pada *Multi-server* Docker Swarm.

Tabel 5. Perintah Deploy Service Pada Docker Swarm

root@node 1
docker stack deploy -c nama-file nama-service

Hasil eksekusi *compose file* dapat dilihat pada gambar 10. *Container* Nginx-PHP akan secara otomatis dibuat kedalam setiap node yang terhubung dalam *multi-server*, hal tersebut dapat terjadi karena *file compose* telah mendefinisikan seluruh *node* dalam *multi-server* untuk mejalankan aplikasi Nginx-PHP. Sedangkan *container* Mysql akan dijalankan pada *node manager*.

```
root@node1:~# docker stack ps web-test
ID             NAME                                    IMAGE                          NODE
zrlngv14myw6   web-test_db.1                           mysql:latest                   node1
4b4l62r2b5j0    \_ web-test_db.1                       mysql:latest                   node3
zge973tkipsj    \_ web-test_db.1                       mysql:latest                   node3
o2vovxfz70c7    \_ web-test_db.1                       mysql:latest                   node2
gv06jlr1eilb    \_ web-test_db.1                       mysql:latest                   node3
ru0n4ij88rxj   web-test_nginx-php.o5gpt4yag7cz2gcbtd757tco3   trafex/php-nginx:latest   node3
xkqq5p9klets   web-test_nginx-php.olxcoepiuqxnyj6bvszfrvut6   trafex/php-nginx:latest   node1
khv2klcbbkas   web-test_nginx-php.r5nsi3p4mfdqjawnujqt9t0cp   trafex/php-nginx:latest   node2
9eaesbt9lnuo    \_ web-test_nginx-php.r5nsi3p4mfdqjawnujqt9t0cp   trafex/php-nginx:latest   node2
```

Gbr 11. Daftar Service Yang Berjalan Docker Swarm

3) Konfigurasi Microk8s

Microk8s memiliki banyak versi yang telah diluncurkan. Versi terbarunya adalah 1.25 pada saat penelitian ini dilakukan, namun ada sebuah *bug* yang membuat versi tersebut tidak dapat mengundang node kedalam *multi-server*. Oleh karena itu penelitian ini menggunakan Microk8s versi 1.23 *stable*.

Sama halnya dengan Docker Swarm, untuk dapat mengundang node kedalam multi-server diperlukan perintah untuk inisialisasi pada setiap node.

Tabel 6 Perintah Inisialisasi Node Kedalam Multi-server

root@node1
microk8s add-node
root@node2, node3
microk8s join 192.168.100.2:25000/(Token yang Diproduksi Microk8s)

Konfigurasi *multi-server* yang berhasil dapat dilihat pada status Microk8s. Sejalan dengan *multi-server* yang berhasil dibuat, *high availability* Microk8s juga akan aktif secara otomatis. Detail mengenai *node* yang telah tergabung dalam *multi-server* dapat dilihat pada gambar

```
wald@node1:~$ kubectl get node -o wide
NAME    STATUS   ROLES    AGE   VERSION
node1   Ready    <none>   86m   v1.23.14-2+55c3aa5b608650
node2   Ready    <none>   12m   v1.23.14-2+55c3aa5b608650
node3   Ready    <none>   10m   v1.23.14-2+55c3aa5b608650
```

Gbr 12. Daftar Node Dalam Multi-server Microk8s

Metallb dapat digunakan untuk menjalankan *load balancing* pada *multi-server* microk8s. IP pada *loadbalancer* juga dapat diakses secara publik yang dapat digunakan sebagai *gateway web server*. Untuk itu perlu diaktifkannya fitur tambahan Metallb.

Microk8s dan Docker Swarm menggunakan *image* yang sama yaitu *trafex/php-nginx* dan *mysql*. Pada Microk8s, digunakan *Dockerfile* untuk membangun Docker *image web server*. Fungsinya adalah memindahkan seluruh *file web* sampel ke dalam direktori */var/www/html* pada *image trafex/php-nginx.*

```
FROM trafex/php-nginx:latest
COPY ./web-server-tugas-akhir/. /var/www/html/.
```

Gbr 13. Konfigurasi Dockerfile Web Server

Controller Deployment pada Kubernetes dapat digunakan untuk membangun *pod* untuk dijalankan pada Microk8s. Keuntungan menggunakan *deployment* adalah dapat menentukan replika *pod* yang telah dibuat. *Image waldrl/nginx-server* merupakan *image* yang telah dibangun dengan Dockerfile. Dengan 3 replikasi yang secara otomatis disebarkan ke *node* yang aktif dalam *multi-server*.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: waldrl/nginx-server:v5
        ports:
        - containerPort: 8080
```

Gbr 14 Source Code Konfigurasi Deployment Web Server

Gambar 14 diatas merupakan konfigurasi *Deployment* dengan nama *Controller nginx*, nama ini nantinya berfungsi sebagai identitas pada saat *Controller* berjalan. Labels App digunakan sebagai identitas aplikasi, meskipun diberi nama nginx namun penamaan akan berbeda karena sistem akan secara otomatis menghasilkan kode tambahan sebagai pembeda pada nama aplikasi. Untuk *image* yang digunakan adalah *custom image* yang dibuat berdasarkan *image trafex/php-nginx*, konfigurasi *custom image* dapat dilihat pada gambar 13 *Port* yang didefinisikan merupakan *port default* dari *image trafex/php-nginx* yaitu 8080.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
  loadBalancerIP: 192.168.100.100
```

Gbr 15. Source Code Konfigurasi Service Web Server

Service dapat digunakan untuk mendefinisikan *port, protocol, target port*, dan IP pada *pod*. Identitas *Service* didefinisikan pada "*metadata name*" dan "*selector app*", perbedaannya *selector* digunakan sebagai identitas kumpulan objek sedangkan *name* menjadi identitas sebuah objek. *Port* didefinisikan untuk dapat menangani lalu lintas data pada aplikasi *web server* dengan mengekspos *port* 8080 menjadi 80. Sedangkan IP *load balancer* menjembatani request oleh *user* ke *web server.*

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - image: mysql
          name: mysql
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: pass
          ports:
            - containerPort: 3306
              name: mysql
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql
      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pv-claim
```

Gbr 16. Source Code Konfigurasi Deployment MySQL

Konfigurasi *deployment* untuk Mysql hampir sama dengan Nginx. *Pod* Mysql hanya memiliki satu replika, nantinya ketika pod mati maka akan langsung dibuat ulang dengan perintah *recreate* yang telah didefinisikan. Konfigurasi *volumes* dapat mensinkronkan data pada mysql dengan data yang tesimpan pada penyimpanan lokal. Image yang digunakan adalah Mysql dengan nama pod mysql. *Env* mendefinisikan akses login terhadap *database* yaitu user dan password. Untuk dapat diakses maka *port* harus didefinisikan dengan *default* Mysql *port* adalah 3306. *Volumemounts* mendifinisikan direktori penyimpanan Mysql pada */var/lib/mysql*.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  selector:
    app: mysql
  ports:
  - port: 3306
    targetPort: 3306
    protocol: TCP
  type: LoadBalancer
  loadBalancerIP: 192.168.100.2
```

Gbr 17. Source Code Konfigurasi Service MySQL

Service Mysql nantinya akan digunakan untuk *web* dapat melakukan koneksi ke database melalui Load balancer IP. Untuk dapat mengakses Mysql dari luar maka *port default* harus diekspos. "*port"* merupakan *port* yang diekspos atau bisa dibilang *port* yang diakses dari luar. Sedangkan "*targetPort*" merupakan *port default* yang didefinisikan pada *container*. *LoadBalancerIP* didefinisikan untuk memastikan IP yang digunakan mengakses Mysql adalah 192.168.100.2.

File konfigurasi yang telah dibuat dapat dieksekusi dengan perintah yang ditunjukan tabel 7. *Pod* dan *service* yang dibuat secara otomatis akan disebarkan kedalam setiap *node* aktif pada *multi-server*, hal ini dapat terjadi karena adanya *loadbalancer* Metallb yang diaktifkan sebelumnya.

Tabel 7. Perintah Deploying File Konfigurasi Microk8s

root@node1
microk8s kubectl apply -f file-konfigurasi

Hasil dari *service* dan *pod* yang telah dieksekusi dapat dilihat secara lengkap pada gambar 18. Pada tabel pertama terlihat *pod* apa saja yang aktif serta dijalankan di *node* mana. Tabel kedua menunjukan *service* yang telah dibuat lengkap dengan IP dan *Port* yang telah didefinisikan didalam *file* konfigurasi. Tabel ke 3 menunjukan *deployment* yang aktif pada kasus ini ada *web* dan *database server*, untuk menghapus *pod* secara permanen maka yang dihapus adalah *deployment*

dari pod yang dituju. Terakhir adalah *replicaset* yang menunjukan jumlah replikasi *pod* yang sedang berjalan.

```
wald@node1:~$ kubectl get all -o wide
NAME                     READY   STATUS    RESTARTS   AGE    IP             NODE    NOMI
pod/mysql-7bdfbb9875-9d45g   1/1     Running   0          100s   10.1.135.1     node3   <none
pod/nginx-5c5797cdd5-7f6l2   1/1     Running   0          54s    10.1.166.130   node1   <none
pod/nginx-5c5797cdd5-72srg   1/1     Running   0          54s    10.1.135.2     node3   <none
pod/nginx-5c5797cdd5-g6hqr   1/1     Running   0          54s    10.1.104.2     node2   <none

NAME                 TYPE           CLUSTER-IP       EXTERNAL-IP       PORT(S)          A
service/kubernetes   ClusterIP      10.152.183.1     <none>            443/TCP          4
service/mysql        LoadBalancer   10.152.183.66    192.168.100.2     3306:32210/TCP   1
service/nginx-service   LoadBalancer   10.152.183.225   192.168.100.100   80:31153/TCP     5

NAME                 READY   UP-TO-DATE   AVAILABLE   AGE    CONTAINERS   IMAGES
deployment.apps/mysql   1/1     1            1           104s   mysql        mysql
deployment.apps/nginx   3/3     3            3           54s    nginx        waldrl/nginx-se

NAME                           DESIRED   CURRENT   READY   AGE    CONTAINERS   IMAGES
replicaset.apps/mysql-7bdfbb9875   1         1         1       103s   mysql        mysql
replicaset.apps/nginx-5c5797cdd5   3         3         3       54s    nginx        waldrl/ng
```

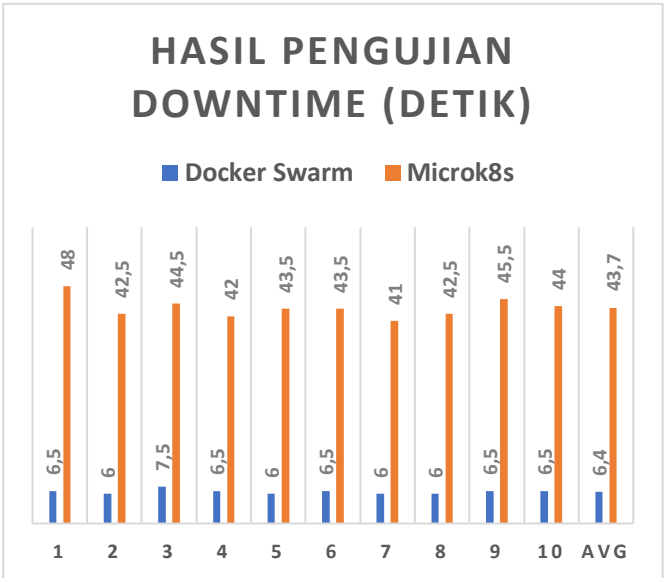
Gbr 18 Daftar Container yang Berjalan Microk8s

B. Pengujian

Pada tahap ini hasil implementasi sistem yang telah dibuat akan dilakukan pengujian. Pengujian dilakukan dengan bantuan perangkat lunak Apache Jmeter untuk melakukan HTTP *Request* ke *server* dan perangkat lunak Systat untuk memonitor penggunaan CPU. Variabel pengujian meliputi *downtime* dan CPU *usage*. Hasil dari pengujian akan dibandingkan untuk mengetahui performa paling efektif antara Docker Swarm dan Microk8s.

1) Pengujian Downtime

Pengujian ini bertujuan untuk mengetahui *downtime* pada saat *server* menangani proses *failover*. Pengujian HTTP request ini dilakukan sebanyak 10 kali, setiap pegujian diberikan waktu 10 menit *request*. *Sample request* yang diberikan selama 10 menit adalah 30000 pada setiap pengujian. Skema pengujian dilakukan dengan mematikan master server, yang mana dalam pengujian ini master server merupakan node 2, sedangkan backup server yang standby selama pengujian berlangsung adalah node 1 dan 3.



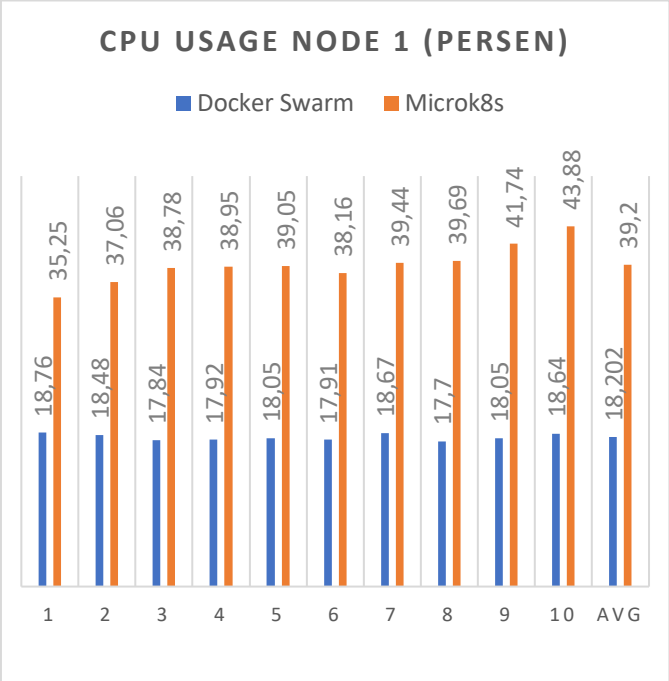
Gbr 19. Grafik Hasil Pengujian Downtime

Dari tabel 4.9 dan gambar 4.35 hasil pengujian dapat dilihat Docker Swarm menghasilkan *downtime* yang lebih rendah dibandingkan dengan Microk8s. Dari 10 pengujian rata-rata waktu hentinya adalah 6,4 detik dengan *downtime* terendah 6 detik dan tertinggi 7,5 detik. Sedangkan pada Microk8s menghasilkan waktu henti yang jauh lebih tinggi dibandingkan dengan Docker Swarm. Rata-rata *downtime* pada *server* dengan Microk8s mencapai 43,7 detik dengan waktu henti terlama mencapai 45,5 detik dan yang tercepat mencapai 41 detik.

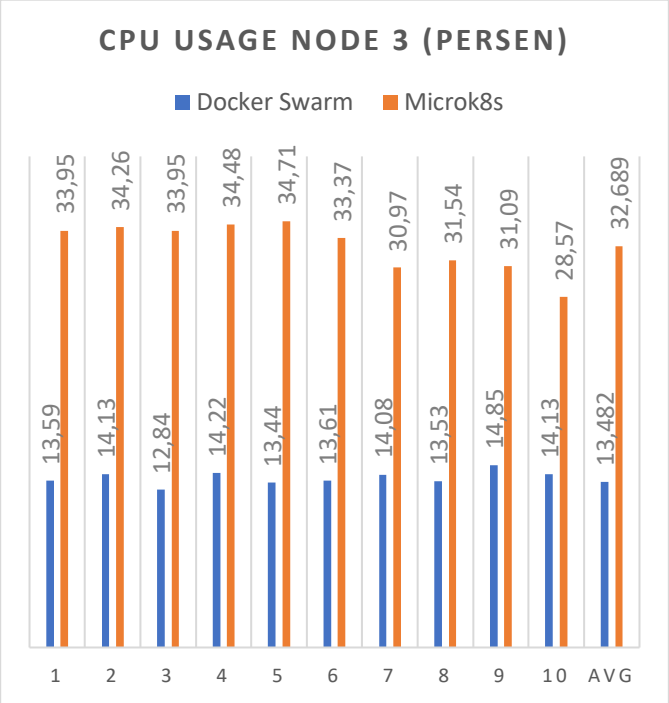
Baik Docker Swarm maupun Microk8s memiliki fitur replikasi dan *load balancer* yang dapat membantu dalam sinkronisasi data *web server*. Berbeda dengan Docker Swarm yang fitur *load balancer* telah otomatis aktif, Microk8s memiliki *plugin load balancer* yang harus diaktifkan terlebih dahulu, *plugin load balancer* yang digunakan adalah Metallb. Sistem replikasi yang membantu dalam proses sinkronisasi *service web server* membuat tingkat ketersediaan *server* menjadi lebih Dalam mengatasi permasalah *downtime* fitur *load balancer* dan Keepalived dapat dengan baik dijalankan meskipun belum seefisien Docker Swarm. *Load balancer* Docker Swarm yang digabungkan dengan sistem *failover* Keepalived memiliki efektifitas yang baik dengan tingkat ketersediaan yang cukup tinggi dan berpengaruh pada hasil *downtime* yang rendah.

2) Pengujian Penggunaan CPU

Pengujian ini bertujuan untuk mengetahui penggunaan sumber daya perangkat keras terutama CPU saat menengani request HTTP pada *web server* yang mengalami *downtime*. Pada tahap ini akan diuji penggunaan CPU saat menangani *Failover*. *Node* yang diamati merupakan *server* yang *standby* sebagai backup *server*, dalam hal ini *node* 2 sebagai *master* yang mana dalam skema pengujian akan dimatikan pada saat proses HTTP *request* dijalankan. *Node* 1 dan *node* 3 menjadi *backup server*, kedua *node* tersebut akan diawasi penggunaan CPU dengan perangkat lunak Systat. Hasil dari pengujian merupakan rata-rata penggunaan CPU selama percobaan berlangsung. Percobaan dilakukan selama 10 menit dengan 30000 *sample request* menggunakan Apache Jmeter.



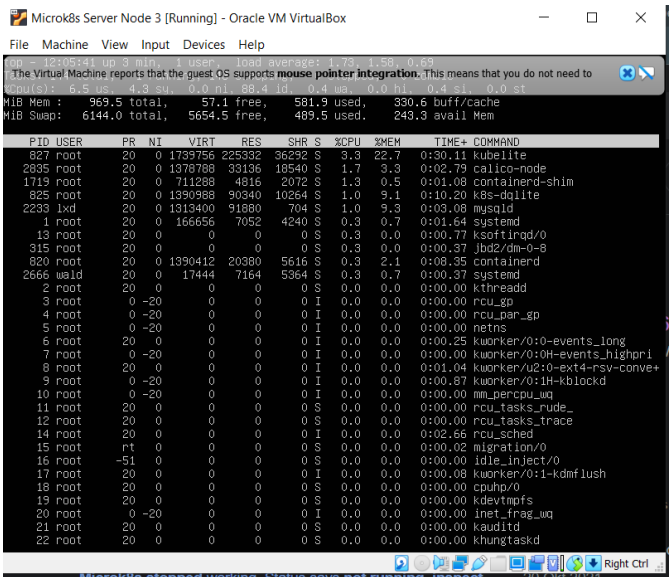
Gbr 20. Grafik Pengujian CPU Usage Node 1



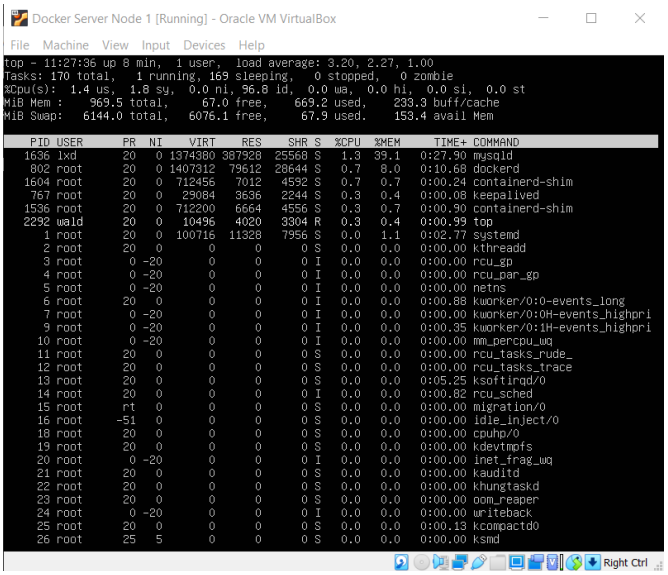
Gbr 21. Grafik Pengujian CPU Usage Node 3

Berdasarkan data yang telah dipaparkan pada tabel dan grafik pengujian CPU *usage* diatas, *load balancer* internal dapat berjalan dengan cukup baik dibuktikan dari penggunaan CPU yang cukup seimbang pada semua *node* yang berjalan. Perbandingan antara kedua *orchestration tool* memiliki selisih penggunaan CPU yang cukup jauh pada node yang aktif saat pengujian dilakukan. Docker Swarm lebih efektif dalam penggunaan CPU dibandingkan dengan Microk8s, dengan selisih lebih dari 2 kali lipat hampir pada keseluruhan hasil pengujian. Berdasarkan hasil penelitian (Sebastian B¨ohm & Guido Wirtz, 2021) dan (Sorensen, 2023) kecenderungan penggunaan CPU pada Microk8s cukup tinggi dibandingkan dengan platform lain seperti K8s dan K3s.

Docker Swarm memiliki fitur dan *plugin* yang minimal, membuat Docker Swarm terasa lebih mudah dioperasikan pada saat implementasi sistem yang dibutuhkan. Hal tersebut juga berpengaruh pada penggunaan sumber daya yang rendah namun dengan hasil yang cukup memuaskan. Sistem replikasi *container* otomatis milik Docker Swarm yang mana jika node *server* mati maka *container* juga dinonaktifkan secara otomatis, hal ini membuat kebutuhan sumber daya CPU juga lebih menjadi lebih rendah pada *node server* yang masih aktif. Microk8s yang basisnya K8S menjadikannya kaya akan fitur yang dibutuhkan dalam membuat sebuah sistem *multi-server* dengan tingkat kompleksitas yang tinggi [11]. Namun hal tersebut juga berpengaruh pada hasil penggunaan sumber daya CPU yang cukup besar. Dapat dilihat pada gambar 22 dan 23 dimana penggunaan CPU yang terpaut cukup jauh dihasilkan dari sistem Micok8s yang mengambil cukup banyak sumber daya dibandingkan dengan container yang berjalan. Sedangkan Docker Swarm menggunakan lebih sedikit sumber daya dikarenakan penggunaan sistem yang berjalan pada Docker Swarm jauh lebih minimal dan efisien.

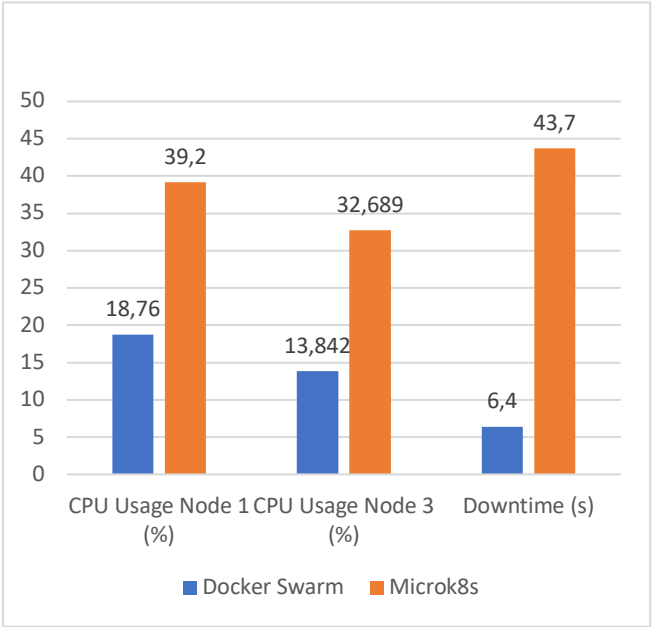


Gbr 22. Penggunaan CPU Mode Stand By Pada Microk8s



Gbr 23. Penggunaan CPU Mode Stand By Pada Microk8s

Berdasarkan gambar 24 dapat dilihat hasil rata pengujian downtime dan penggunaan CPU. Dari hasil tersebut dapat dilhat korelasi antara penggunaan CPU dengan waktu henti yang terjadi. Penggunaan CPU yang tinggi dapat memengaruhi *server* dalam menangani *request* yang menyebabkan terjadinya peningkatan waktu henti. Hal tersebut dapat dibuktikan dengan pengunaan CPU yang mencapai 100% pada beberapa waktu ketika skema failover dijalankan. *Downtime* yang dihasilkan pada Docker Swarm jauh lebih rendah dibandingkan dengan Microk8s, yang mana hal ini juga bisa dipengaruhi oleh penggunaan sumber daya yang cukup besar pada Micok8s. Dengan memerhatikan faktor lain seperti arsitektur orkestrasi serta sistem yang berjalan pada masing-masing *platform*, penggunaan sumber daya CPU juga dapat memengaruhi hasil *downtime*. Hasil tersebut dapat dijadikan sebagai bahan acuan untuk memilih *patform* mana yang sesuai dengan kebutuhan *server* yang akan digunakan. Untuk *server* dengan konfigurasi yang sederhana dengan tingkat kebutuhan sumber daya yang minimal maka Docker Swarm akan menjadi jawaban yang relevan. Sedangkan untuk Microk8s dapat digunakan sebagai alternatif *orchestration tool* untuk kebutuhan *server* yang lebih kompleks dengan sumber daya yang lebih besar.



Gbr 24. Grafik Perbandingan Rata-rata Hasil Pengujian

IV. KESIMPULAN

Dari penelitian ini penulis dapat menerapkan sistem *high availability* dengan menggunakan *orchestration tool* Docker Swarm dan Microk8s. Hasilnya Hasil pegujian memiliki tingkat kestabilan dalam menangani system *failover* dengan cukup baik, dibuktikan dari lonjakan waktu henti yang tidak terpaut jauh pada setiap pengujian. Namun dapat dilihat dari grafik hasil pengujian jarak antara Docker Swarm dengan Microk8s terpaut cukup dalam menangani permasalahan *downtime*. Hal ini dapat terjadi karena berbagai hal, salah satunya adalah kurang efektifnya penggunaan CPU pada sistem Microk8s. *Plugin* yang dijalankan pada Microk8s cukup banyak memakan ruang pada CPU membuat saat *request* terjadi tidak dapat diatasi secara efesien. Dengan argumen dan hasil telah dipaparkan Docker Swarm memiliki performa yang lebih baik dalam menangani masalah downtime serta lebih efisien dalam penggunaan CPU.

V. SARAN

Berdasarkan hasil dari penelitian ini,peneliti memberikan saran untuk penelitian serupa kedepannya. Implementasi dapat menggunakan actual server baik menggunakan *platform cloud* ataupun lokal *server*. Percobaan dapat dilakukan dengan distribusi Kubernetes lainnya seperti K8S, Minikube, K3S, K3D. Variabel pengujian dapat ditingkatkan seperti penambahan perbandingan penggunaan sumber daya lainnya seperti RAM. Skema pengujian high availability dapat divariasi tidak hanya dengan mematikan dan menyalakan ulang server, seperti serangan malware atau DDOS.

UCAPAN TERIMA KASIH

Penulis senantiasa mengucap syukur kepada Tuhan YME atas segala berkah, rahmat dan pertolongannya, sehingga penulis mampu menyelesaikan proyek dan artikel ilmiah ini dengan baik. Terimakasih penulis haturkan kepada kedua orangtua dan saudara yang selalu memberi semangat dan dukungan, dosen pembimbing skripsi yang selalu memberikan masukan dan saran yang membangun kepada penulis, sahabat dan teman yang selalu memberikan dukungan dan dorongan dalam melakukan penelitian. Terimakasihkepada diri sendiri karena dapat berkompromi untuk menggapai tujuan yang ingin dicapai.

REFERENSI

[1] Jafaruddin Gusti Amri Ginting, Syariful Ikhwan, Muhammad Naufal Ammar, "Analisis Performansi High Availability Web Server Pada Cluster GKE (Google Kubernetes Engine) Menggunakan Infrastruktur Google Cloud Platform," *Jurnal Nasional Informatika Dan Teknologi Jaringan- VOL. 5 No.2 (2021) Edisi Maret,* pp. 348-354, 2021.

[2] Supriyadi, , Didik Setiyadi, "Virtualisasi Server Failover Clustering Menggunakan Network Development Life Cycle Di SMK Negeri 1 Kota Bekasi," *JURNAL MAHASISWA BINA INSANI Vol.4 No.2,* p. 115 – 124, 2020.

[3] Arief Husaini, Umar Ali Ahmad, R.Rogers Dwiputra Setiady, "Implementasi High Availability Dengan Metode Failover Pada Amazon Web Service," *e-Proceeding of Engineering : Vol.8, No.6,* pp. 12007-12013, 2021.

[4] Chrisna Fiddin, Dr. Rendy Munadi, Ir.,M.T. , Ratna Mayasari, S.T, M.T., "Analisis Performansi Virtualisasi Container Menggunakan Docker Dibawah Serangan Networked Denial Of Service," . *e-Proceeding of Engineering : Vol.5, No.1,* pp. 281-290, 2018.

[5] Ali Akbar Khatami, Yudha Purwanto, Muhammad Faris Ruriawan, "IMPLEMENTASI HIGH AVAILABILITY STORAGE SERVER," *International Conference on Information Technology Systems and Innovation (ICITSI),* pp. 74-78, 2020.

[6] Bayu Arifat Firdaus, Vera Suryani, Siti Amatullah Karimah, " Analisis Performansi Proses Scaling Pada Kubernetes Dan Docker Swarm Menggunakan Metode Horizontal Scaler," *e-Proceeding of Engineering : Vol.7, No.2,* pp. 7793-7808., 2020.

[7] Sebastian B¨ohm & Guido Wirtz, "Profiling Lightweight Container Platforms: MicroK8s and K3s in Comparison to Kubernetes," in *J. Manner, S. Haarmann, S. Kolb, N. Herzberg, O. Kopp (Eds.): 13th ZEUS Workshop, ZEUS 2021, Bamberg, held virtually due to Covid-19 pandemic, Germany*, Bamberg, 2021.

[8] Stefanus Eko Prasetyo, Yulfan Salimin, "Analisis Perbandingan Performa Web Server Docker Swarm dengan Kubernetes Cluster," *Conference on Management, Business, Innovation, Education and Social Science Vol. 1 No. 1,* pp. 825-833, 2021.

[9] Min HAN, Deng-guo YAO, and Xiao-lin YU, "A Solution for Instant Response of Cloud Platform Based on Nginx + Keepalived," *International Conference on Computer Science, Communications and Multimedia Engineering (CSCME),* pp. 1-8, 2019.

[10] R. Sorensen, "An evaluation of edge deployment models for Kubernetes," *Degree Project in Computing Science Engineering, 30 ECTS,* pp. 1-30, 2023.

[11] Anggi Hanafiah, Rizky Wandri, "Implementasi Load Balancing Dengan Algoritma PenjadwalanWeighted Round Robin Dalam Mengatasi Beban Webserver," *Conference on Management, Business, Innovation, Education and Social Science,* pp. 825-833, 2021.