Penerapan *Container Load Balancing* untuk Manajemen Trafik pada *Learning Management System*

Tegar Sukma Hendrana[1], I Made Suartana[2]

[1,2] Jurusan Teknik Informatika, Fakultas Teknik, Universitas Negeri Surabaya
[1]tegar.18074@mhs.unesa.ac.id
[2]imadesuartana@unesa.ac.id

***Abstrak*— Metode pembelajaran daring berbasis *e-learning* saat ini menjadi keharusan untuk diterapkan di berbagai lembaga pendidikan di Indonesia dalam rangka menunjang kegiatan pembelajaran yang lebih fleksibel. Kondisi tersebut menjadikan Moodle *Learning Management System* (LMS) sebagai salah satu platform yang banyak digunakan untuk merealisasikan sistem pembelajaran berbasis *e-learning*.**

Peningkatan intensitas penggunaan serta kompleksitas fitur dari Moodle LMS yang mencakup banyak aktivitas didalamnya, menjadikan standar *availability server* menjadi komponen penting untuk menunjang aktivitas Moodle yang optimal. Di sisi lain, arsitektur server web tunggal sudah tidak lagi relevan dengan kondisi tersebut. Oleh karena itu, membangun infrastruktur server Moodle yang mampu menunjang berjalannya aktivitas pembelajaran dengan baik sangatlah penting. Penerapan arsitektur *clustering web server* berbasis *docker swarm cluster* yang memuat implementasi metode *container load balancing* menjadi solusi mengenai isu standar *availability server* Moodle. *Docker swarm cluster* terdiri dari *node manager* dan *node worker*, dengan implementasi *container load balancing*, *node manager* mampu mengarahkan *user request* menuju *node worker* secara merata sehingga *load traffic* dapat diatasi dengan baik dan kinerja Moodle lebih optimal.

Dari penelitian ini, diperoleh hasil bahwa server Moodle yang dibangun di lingkungan *docker swarm cluster* dengan menerapkan metode *container load balancing* mampu merespon *load traffic* dari *user request* dengan baik dilihat dari hasil pengujian dengan parameter *throughput* dan *error rate* pada aktivitas *login, view course, assignments* dan *quiz*. NFS mampu menyediakan penyimpanan data persisten yang dibutuhkan Moodle dalam melakukan tracking terhadap volume *moodlecode* dan *moodledata.* Mekanisme *scaling* dan *failover* dari *docker swarm* berjalan dengan baik sehingga membuat tingkat ketersediaan server tinggi.

***Kata Kunci*— Learning Management System, Moodle, Availability Server, Clustering Web Server, Container, Docker Swarm, Load Balancing**

I. PENDAHULUAN

Perkembangan teknologi informasi yang begitu pesat serta diiringi dengan munculnya pandemi *covid-19* menjadikan sistem pembelajaran berbasis *e-learning* diterapkan di semua lembaga pendidikan di Indonesia. Kondisi tersebut menjadi faktor meningkatnya penggunaan media pembelajaran *e-learning* yang dikenal dengan *learning management system* (LMS) untuk menunjang proses kegiatan belajar mengajar di lembaga pendidikan. LMS merupakan perangkat lunak atau software yang digunakan untuk keperluan administrasi, dokumentasi, pencarian materi, laporan kegiatan, pemberian materi pelatihan kegiatan belajar mengajar dan mengelola kegiatan hasil pembelajaran secara online [9]. Dengan beragam fungsi tersebut, tenaga pendidik bisa lebih mudah dalam mengelola materi pembelajaran dan melakukan monitoring aktivitas belajar peserta didik dari mana saja dan kapan saja. Menurut [9], Penerapan media *e-learning* berbasis *learning management system* memenuhi kategori efektivitas karena dapat meningkatkan kognitif mahasiswa dan aktivitas mahasiswa.

Saat ini tersedia banyak platform LMS yang bisa digunakan, salah satu platform yang cukup populer adalah Moodle *(Modular Object Oriented Dynamic Learning Environment)* yang merupakan platform LMS berbasis website dan bersifat *open source*. Saat ini sebagian besar lembaga pendidikan di Indonesia memanfaatkan Moodle sebagai platform LMS untuk menunjang proses belajar mengajar. Berdasarkan Moodle *statistics* yang dipublikasi di website resmi Moodle, hingga pada saat penelitian ini dibuat, Indonesia masuk di peringkat 7 di antara 244 negara dalam kategori registrasi Moodle terbanyak. Kondisi tersebut menjadi bukti bahwa Moodle menjadi platform LMS yang memiliki kualitas baik dan memiliki tingkat kepercayaan tinggi dari pengguna, selain itu Moodle yang bersifat *open source* juga menjadi pertimbangan karena memudahkan dalam proses pengembangan media pembelajaran menyesuaikan dengan kebutuhan tiap sekolah maupun perguruan tinggi. Aktivitas pembelajaran yang bisa dilakukan pada Moodle hingga saat ini sangat beragam, mulai dari *assignments, chat, choice, database, feedback, forum, glossary, h5p activity, lesson,* (LTI) *external tool, quiz, scorm, survey, wiki,* dan *workshop* [1]. Jika ditinjau dari segi arsitekturnya, Moodle dibangun menggunakan basis bahasa pemrograman PHP *(Hypertext Preprocessor)* dan menggunakan database berbasis SQL *(Structured Query Language)*.

Ditengah fenomena peningkatan penggunaan Moodle LMS serta fungsionalitas dari Moodle LMS yang mencakup banyak aktivitas didalamnya, tingkat ketersediaan server menjadi faktor penting bagi berjalannya aktivitas Moodle. Di sisi lain, arsitektur server web tunggal yang dianggap tidak memiliki tingkat ketersediaan yang tinggi untuk menunjang kinerja dari Moodle dalam mengatasi load traffic di setiap aktivitas yang dilakukan pengguna. Menurut [3], Peningkatan akses menuju sumber daya *e-learning* secara terus-menerus dapat menimbulkan banyak masalah termasuk isu ketersediaan sumber daya server, karena itu kebutuhan untuk terus meningkatkan dan memperluas sumber daya server juga penting. Berdasarkan kondisi tersebut, penerapan metode deployment dan arsitektur server yang relevan serta memiliki standar availability sangat diperlukan untuk memastikan

layanan Moodle bisa berjalan dengan optimal dalam menunjang aktivitas pembelajaran.

Pertimbangan penerapan metode *clustering server* menjadi salah satu solusi untuk memenuhi standar availability server Moodle. *Clustering server* merupakan metode deployment suatu aplikasi dengan mereplikasi server menjadi beberapa bagian atau node untuk dijalankan sebagai satu entitas dalam rangka menjamin tingkat *availability server* ketika terjadi suatu masalah pada salah satu server maupun ketika terjadi lonjakan akses ke server. Teknologi yang mendukung penerapan dari metode *clustering server* saat ini adalah kontainerisasi (virtualisasi berbasis kontainer). *Container* merupakan *lightweight virtualization* yang dapat bekerja secara langsung didalam *hos*t sistem operasi dan menjalankan segala proses instruksi secara langsung kepada *core* CPU [7]. *Container* memiliki tingkat independensi yang cukup tinggi, sehingga memungkinkan aplikasi yang diisolasi bisa dijalankan di infrastruktur manapun.

Dalam penerapannya, kontainerisasi membutuhkan suatu *tools* yang digunakan untuk membuat dan mengelola *container,* dan tools yang paling popular saat ini adalah docker. *Docker* merupakan platform *open source* yang memberi kemudahan bagi developer untuk mengembangkan, mengirimkan, dan menjalankan aplikasi. Menurut [7], Desain arsitektur docker memudahkan dalam proses distribusi dan pengembangan aplikasi secara lebih cepat karena docker bersifat *lightweight containerization* serta memiliki beragam komponen dan fitur yang mampu memudahkan developer untuk mengembangkan dan memantau kinerja aplikasinya dengan mudah. *Docker* mempunyai tingkat portabilitas yang tinggi dari segi *uptime* dan *deployment* aplikasi, karena proses pembuatan *container* hanya membutuhkan waktu beberapa detik saja sampai *container* berjalan dan bisa memberikan layanan *(uptime)*, selain itu docker juga memiliki *online repository* yang bernama *docker hub*, *repository* tersebut bisa menyimpan image dari komponen aplikasi untuk nantinya dapat di *pull* dan diaplikasikan kedalam *container* sehingga memudahkan proses *production*.

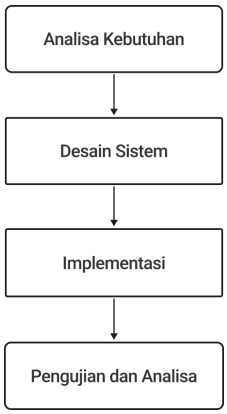
Untuk mengelola banyak *container* dalam beberapa *node* pada *cluster server* bukanlah hal yang mudah, diperlukan suatu *orchestrator* yang mampu mengelola *container* dalam tiap *node* agar bisa saling terhubung satu sama lain. *Docker* mempunyai mode yang bernama *docker swarm*. *Docker swarm* merupakan salah satu *docker container orchestrator* yang memungkinkan kita untuk memanajemen *container* kedalam lingkungan *cluster* yang dibangun menggunakan *docker. Docker swarm cluster* memiliki dua tipe node, yaitu *node manager* dan *node worker*. Menurut [11] *Docker swarm cluster* memungkinkan administrator untuk menambahkan jumlah *node* yang tidak terbatas dan memungkinkan untuk menjalankan *container* dengan jumlah banyak pada *node*. Proses *deployment* pada mode *docker swarm* memiliki tingkat *scalability* yang tinggi, aplikasi dapat diskalakan secara horizontal menjadi beberapa server dalam level *container*, dan direplikasi menjadi beberapa *node* sehingga apabila server yang lain mengalami masalah masih ada server lain yang bisa melayani *request*. Sesuai dengan konsep dasar, dimana *node manager* akan mengarahkan permintaan dari *client* menuju *node worker* dengan menggunakan mekanisme *load balancing* internal yang dimiliki *docker swarm*. Mekanisme *load balancing* internal *docker swarm* berfokus dalam mendistribusikan permintaan kepada *node worker* secara seimbang berdasarkan permintaan pengguna.

Load balancing menjadi isu yang menarik dalam implementasi *docker swarm cluster* ini, Menurut [6], *Docker swarm* memiliki fitur *load balancing* internal yang bernama *ingress load balancing*, namun hanya mengatur antar container di dalam *host* dan tidak dapat di monitor. Di sisi lain, dalam upaya manajemen trafik pada server web, monitoring *load traffic* menjadi hal yang sangat penting untuk memastikan *request client* berhasil didistribusikan dengan baik. Sehingga dibutuhkan konfigurasi tambahan dan *tools load balancer* untuk memastikan *load balancing* berjalan dengan baik dan bisa dimonitor.

Sehingga pada penelitian ini, penerapan arsitektur server berbasis *docker swarm cluster* untuk *deployment* Moodle dengan menerapkan metode *container load balancing* diharapkan mampu mengatasi *load traffic* dari *request client* dengan baik serta dapat terdistribusi secara merata menuju server Moodle yang ada pada klaster, sehingga kinerja Moodle dan tingkat ketersediaan server Moodle menjadi lebih optimal.

II. METODOLOGI PENELITIAN

Metode yang digunakan pada penelitian ini adalah metode *experimental design*. Dalam upaya melakukan penerapan teknik *load balancing* untuk Moodle LMS yang di *deploy* pada lingkungan infrastruktur server berbasis *docker swarm cluster* dan untuk mengetahui data kinerja dari Moodle LMS ketika merespon *request client* dengan parameter pengujian yang ditentukan, maka beberapa tahapan dilakukan untuk mencapai tujuan tersebut. Berikut ini merupakan tahapan yang dilakukan dalam penelitian ini:



Gbr. 1 Alur Penelitian.

A. Analisa Kebutuhan

Tahap pertama yang dilakukan dalam penelitian ini adalah melakukan analisis terhadap kebutuhan yang diperlukan dalam menunjang proses penelitian. Analisis kebutuhan dibagi menjadi beberapa bagian, yaitu:

a. Kebutuhan Data

Data yang digunakan dalam penelitian ini diambil daribeberapa sumber referensi, yaitu:

1. *Studi Literatur*
   Penelitian ini mengambil referensi dari jurnal nasional, jurnal internasional, laporan, makalah, serta artikel dari internet yang relevan dengan implementasi model *deployment container* pada *cluster docker swarm* yang sesuai dengan topik penelitian ini.
2. *Observasi*
   Penelitian ini juga melakukan observasi dengan mengunjungi situs referensi mengenai swarm cluster, serta kolom komentar dari platform *learning management system* untuk mengetahui *pain poin* dari pengguna selama menggunakan platform Moodle *learning management system*.

b. Kebutuhan Fungsional Sistem

Kebutuhan fungional dari mekanisme *load balancing* Moodle server pada *docker swarm cluster* yaitu:

1. *Docker swarm* mampu menjalankan mekanisme *failover* jika salah satu *host node* mati.
2. Mampu menyediakan *Moodle volume* yang bisa diakses oleh *node* yang berisi *Moodle container*.
3. Mampu menyediakan *Moodle server* dan menampilkan halaman Moodle.
4. Pendistribusian *request* oleh *docker swarm* berjalan dengan baik.
5. Moodle service dapat di *scaling* sesuai ketentuan maksimal yang didefinisikan di vagrantfile.

c. Kebutuhan Perangkat

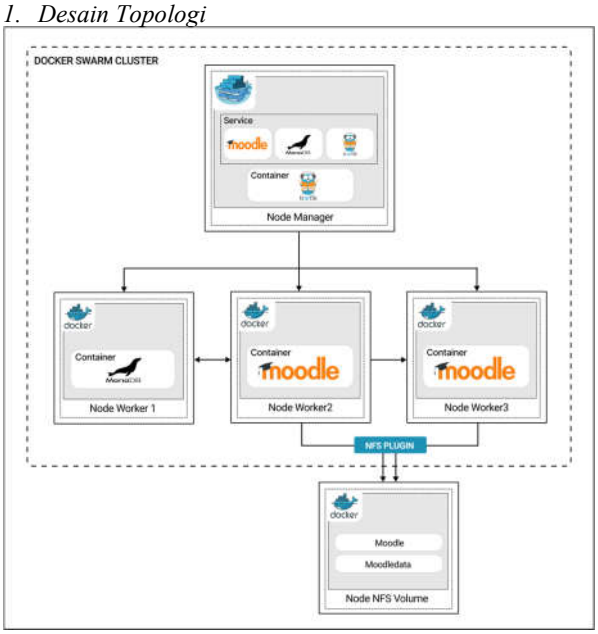
Spesifikasi perangkat yang digunakan dalam penelitian ini terbagi kedalam dua jenis, yaitu:

1. Perangkat Host:
   a. Laptop Lenovo Thinkpad T440P
   b. Processor Intel Core i5 Gen-4
   c. RAM 8 GB
   d. SSD 250 GB
   e. HDD 500 GB
   f. Sistem Operasi Windows 10 64-bit
   g. Vagrant 2.2.19
   h. VitualBox 6.1
   i. Moba Xterm
   j. Apache Jmeter
2. Perangkat Virtual Server
   a. Sistem Operasi Ubuntu 20.04
   b. 2 vCPU
   c. RAM 1 GB
   d. Memori 8 GB
   e. Docker v20.10.17
   f. Moodle 3.9
   g. MariaDB Latest
   h. Traefik
   i. NFS

B. Desain Sistem

Sistem yang dirancang dalam penelitian ini menggunakan empat *instances* yang berada dalam *cluster docker swarm* dan satu *instances* diluar *cluster docker swarm* karena bertindak sebagai NFS server. *Container* yang berada dalam beberapa *node* terhubung dalam satu entitas *cluster*. Untuk kebutuhan penyimpanan data *persistent* agar *docker volume* bisa diakses oleh *node* dalam *cluster,* maka dibuat satu media *volume sharing* untuk menempatkan file yang nantinya diakses *container* dalam *node cluster.* Mekanisme *container load balancing* memanfaatkan traefik sebagai tools *load balancer* modern yang memiliki tingkat konfiguasi cukup sederhana. Berikut merupakan desain sistem dalam penelitian ini:

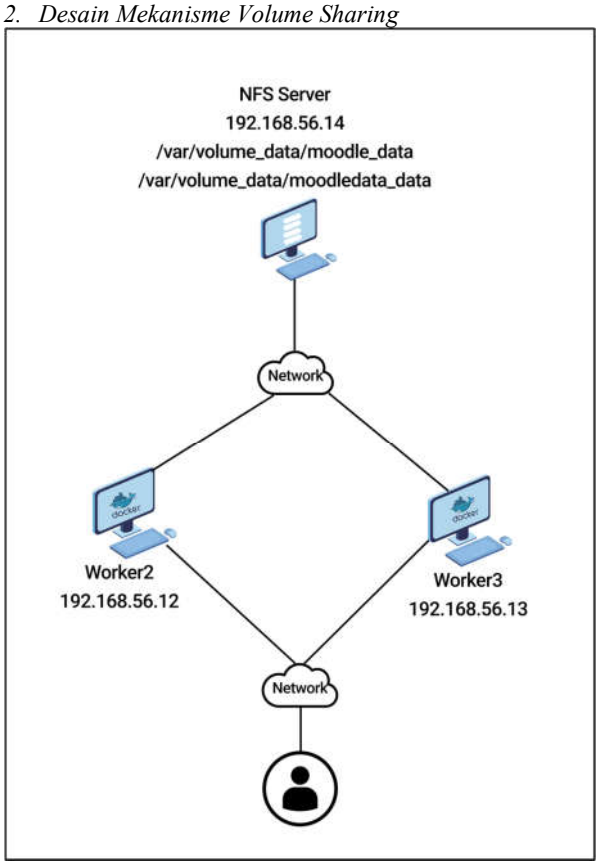
1. Desain Topologi



Gbr. 2 Desain Topologi Sistem.

Penelitian ini menggunakan empat node yang tergabung dalam *cluster swarm*, dimana tiap node berjalan di *instances* ubuntu. *Node* pertama sebagai *swarm manager* sekaligus *worker*, *node* ini berisi *service* Moodle, mariadb dan traefik, selain itu *node* ini juga menjalankan *container* traefik. *Node* kedua sebagai *worker 1*, *node* ini menjalankan *container* mariadb. *Node* ketiga dan keempat sebagai *swarm worker* dan *nfs client*, *node* ini menjalankan *container* Moodle. Kemudian satu *instances* diluar *cluster swarm* sebagai nfs-server yang menampung *volume moodle data* dan *moodle code* untuk memastikan sinkronisasi file yang ada dalam volume tersebut.

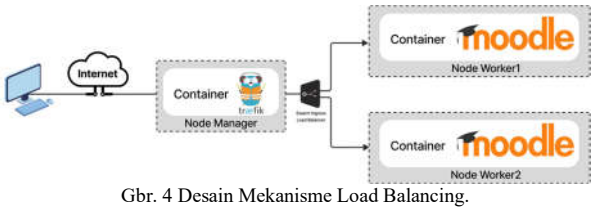
2. *Desain Mekanisme Volume Sharing*



Gbr. 3 Desain Mekanisme Volume Sharing.

Perancangan NFS *volume* sebagai penyimpanan *moodlecode* dan *moodledata* menggunakan *instance nfs server* dengan IP 192.168.56.14. NFS server membuat direktori */var/volume_data* yang diekspor ke jaringan agar bisa diakses oleh *node worker* sebagai *volume*. Didalam */var/volume_data* terdapat dua direktori yaitu */moodle_data* dan */moodledata_data* yang masing-masing digunakan untuk menyimpan data dari *moodlecode* dan *moodledata* ke volume lokal.

3. *Desain Mekanisme Load Balancing*



Gbr. 4 Desain Mekanisme Load Balancing.

Load balancing melibatkan tiga *node*, yaitu *node manager* yang menjalankan *container* traefik, *node worker 2* dan *node worker 3* yang menjalankan container Moodle. *Load balancer* pada penelitian ini menggunakan traefik yang berada pada *node manager* dengan ip 192.168.56.10 yang memproses *request client* yang masuk dan meneruskannya ke *node worker* di lingkungan *docker swarm cluster*. Pendefinisian *load balancing* dituliskan pada *stackfile*.

4. *Desain Arsitektur Instances Virtual Server*

Tabel 1. Arsitektur Instances Virtual Machine..

Node Manager	
CPU	2 CPU
Memory	1 GB
Storage	8 GB
Sistem Operasi	Ubuntu 20.04
Roles	Swarm Manager
Container	Traefik
IP Address	192.168.56.10
Node Worker1	
CPU	2 CPU
Memory	1 GB
Storage	8 GB
Sistem Operasi	Ubuntu 20.04
Roles	Swarm Worker
Container	MariaDB
IP Address	192.168.56.11
Node Worker2	
CPU	2 CPU
Memory	1 GB
Storage	8 GB
Sistem Operasi	Ubuntu 20.04
Roles	Swarm Worker NFS Client
Container	Moodle
IP Address	192.168.56.12
Node Worker3	
CPU	2 CPU
Memory	1 GB
Storage	8 GB
Sistem Operasi	Ubuntu 20.04
Roles	Swarm Worker NFS Client
Container	Moodle
IP Address	192.168.56.13
NFS Server	
CPU	2 CPU

Memory	1 GB
Storage	8 GB
Sistem Operasi	Ubuntu 20.04
Roles	NFS Server
Container	192.168.56.14

Penelitian ini secara keseluruhan menggunakan lima *instances virtual machine*, dimana empat *instances* berada dalam satu entitas *cluster* yang dikoordinasikan menggunakan mode *docker swarm*, dengan rincian satu *node* berperan sebagai *manager* dan *node* lain berperan sebagai *worker*. Kemudian dan satu *instances* diluar *cluster swarm* sebagai nfs-server

C. Implementasi

Setelah menganalisa kebutuhan dan melakukan perancangan sistem, tahap selanjutnya adalah mengaplikasikan hasil rancangan yang telah dibuat untuk pengembangan Moodle pada arsitektur *docker swarm cluster* dengan penerapan metode *container load balancing*. Implementasi yang dilakukan meliputi *development instances virtual server,* implementasi *docker swarm,* implementasi NFS *volume sharing,* implementasi *service,* implementasi *load balancing,* dan *deployment system*.

D. Pengujian dan Analisa

Pengujian yang diterapkan pada penelitian ini meliputi pengujian kinerja, pengujian *scaling,* dan pengujian fungsional. Pengujain tersebut dilakukan sebagai upaya untuk mengetahui kinerja moodle, serta mekanisme *scaling* dan mekanisme *failover* pada *swarm cluster*. Berikut merupakan penjelasan pada tiap pengujian, yaitu:

a. Pengujian Kinerja

Pengujian kinerja sistem ini dilakukan dengan memberikan *load traffic* dari *request user* dengan jumlah 50 user, 100 user, 150 user dan 200 user. Pemberian *load traffic* akan dilakukan menggunakan software Jmeter pada saat aktivitas *login*, *view course, assignment* dan *quiz*. Hasil pengujian secara otomatis disimpan di fitur *summary report* dengan parameter nilai *throughput* dan *error rate.*

b. Pengujian Scaling

Pengujian *scaling* dilakukan untuk memastikan *docker swarm* mampu melakukan replikasi terhadap Moodle ketika *service* sedang berjalan. Replikasi dilakukan saat jumlah service awal satu replika kemudian dilakukan scaling menjadi dua replika dan empat replika..

c. Pengujian Fungsional

Pengujian fungsional dilakukan untuk menguji apakah sistem secara fungsional lingkungan server moodle sudah berjalan dengan baik dan benar. Pengujian dilakukan dengan menjalankan interaksi antara *node manager* dan *node worker*. Pengujian dilakukan untuk memastikan *docker swarm* dapat menjalankan mekanisme *failover* apabila salah satu host dalam klaster mati.

Setelah pengujian dilakukan, akan diperoleh data hasil pengujian yang kemudian dilakukan Analisa menggunakan parameter yang ditentukan untuk mengetahui kualitas hasil pengujian.

III. HASIL DAN PEMBAHASAN

Setelah melakukan perancangan arsitektur sistem dan pengambilan data dari tahap pengujian performa Moodle dalam berbagai aktivitas dan parameter pengujian yang sudah ditetapkan, maka pada bagian hasil dan pembahasan ini akan menampilan proses konfigurasi arsitektur server dan menampilkan data hasil pengujian dalam bentuk tabel dan narasi pembahasan terhadap data yang dihasilkan

A. Implementasi Sistem

Pada bagian ini akan dibahas mengenai langkah-langkah pengembangan lingkungan *docker swarm cluster* sebagai media *deployment* Moodle, implementasi metode *container load balancing* menggunakan traefik, dan sharing volume menggunakan NFS. Implementasi ini merupakan proses realisasi dari desain sistem yang telah dirancang sebelumnya, berikut ini tahapan-tahapan pada fase implementasi.

1.) Development Instances Virtual Server

Dalam membangun *instances virtual server*, pada penelitian ini memanfaatkan *vagrant* sebagai *software* otomasi yang berfungsi untuk mengefisiensi proses pembuatan *instances*, penentuan spesifikasi *instances* sepenuhnya didefinisikan di *vagrantfile* beserta *provisioning* kebutuhan sistem.

```
servers=[
  {
    :hostname => "manager",
    :ip => "192.168.56.10",
    :box => "aspyatkin/ubuntu-20.04-server",
    :ram => 1024,
    :cpu => 2
  },
  {
    :hostname => "worker-1",
    :ip => "192.168.56.11",
    :box => "aspyatkin/ubuntu-20.04-server",
    :ram => 1024,
    :cpu => 2
  },
  {
    :hostname => "worker-2",
    :ip => "192.168.56.12",
    :box => "aspyatkin/ubuntu-20.04-server",
    :ram => 1024,
    :cpu => 2
  },
  {
    :hostname => "worker-3",
    :ip => "192.168.56.13",
    :box => "aspyatkin/ubuntu-20.04-server",
    :ram => 1024,
    :cpu => 2
  },
  {
    :hostname => "nfs-server",
    :ip => "192.168.56.14",
    :box => "aspyatkin/ubuntu-20.04-server",
    :ram => 1024,
    :cpu => 2
  }
]
```

Gbr. 5 Source code vagrantfile pendefinisian spesifikasi instances

```
Vagrant.configure(2) do |config|
  servers.each do |machine|
    config.vm.define machine[:hostname] do |node|
      node.vm.box = machine[:box]
      node.vm.hostname = machine[:hostname]
      node.vm.network "private_network", ip: machine[:ip]
      if machine[:hostname] == "manager"
        node.vm.provision "docker",
          images: ["tegarshndrn/moodle:port-8181"]
      else
        node.vm.provision "docker"
      end
      node.vm.provider "virtualbox" do |vb|
        #vb.gui = true
        vb.customize ["modifyvm", :id, "--memory", machine[:ram]]
      end
    end
  end
end
```

Gbr. 6 Source code vagrantfile provisioning instances

Vagrantfile tersebut sebagai bentuk implemantasi dari rancangan *instances virtual server* yang telah dibuat. *Command* pada *vagrantfile* ini mendefinisikan *instances* dalam format stack dengan segala konfigurasinya yang dibungkus dalam *vagrant box*. *Provisioning instances* yang diperlukan untuk pengembangan sistem juga didefinisikan di *vagrantfile*, kebutuhan sistem pendukung tersebut meliputi docker dan Moodle yang secara otomatis ditambahkan pada *instances*.

```
vagrant@manager:~$ docker version
Client: Docker Engine - Community
 Version:           20.10.17
 API version:       1.41
 Go version:        go1.17.11
 Git commit:        100c701
 Built:             Mon Jun  6 23:02:57 2022
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.17
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.17.11
  Git commit:       a89b842
  Built:            Mon Jun  6 23:01:03 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.7
  GitCommit:        0197261a30bf81f1ee8e6a4dd2dea0ef95d67ccb
 runc:
  Version:          1.1.3
  GitCommit:        v1.1.3-0-g6724737
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

Gbr. 7 Versi docker pada instances

Docker yang digunakan pada penelitian ini menggunakan versi terbaru yaitu 20.10.17.

```
vagrant@manager:~$ docker image ls
REPOSITORY              TAG                 IMAGE ID       CREATED
tegarshndrn/moodle      port-8181           978bb9c261e3   2 hours ago
traefik                 <none>              de1a7c9d5d63   12 months ago
adji21/moodle           port-8181           91193e32da2d   13 months ago
bitnami/moodle          3.9.1-debian-10-r16 d20dd0c52777   2 years ago
```

Gbr. 8 Provisioning docker image

Ketersediaan *docker image* pada *instances* diperlukan agar satat pembuatan *container* tidak menyita waktu terlalu lama karena tidak lagi melakukan *download image* dari *docker registry.*

2.) Implementasi Docker Swarm

Implementasi docker swarm untuk membuat lingkungan *cluster swarm* dilakukan pada empat *instances, node manager, node worker-1, node worker-2,* dan *node worker-3*. Inisialisasi *docker swarm* dilakukan di *node manager.*

Tabel 3. Inisialisasi docker swarm

vagrant@manager
docker swarm init
vagrant@worker-1, vagrant@worker-2, vagrant@worker-3
docker swarm join-token

Untuk memastikan lingkungan *docker swarm cluster* berhasil dibangun, dilakukan pengecekan menggunakan perintan *docker node ls.*

```
vagrant@manager:~$ docker node ls
ID                            HOSTNAME   STATUS   AVAILABILITY   MANAGER
evq805jkpr1ud6sriy5h9qkan *   manager    Ready    Active         Leader
yphtj6eoqydq041voq4qs5kmt     worker-1   Ready    Active
sl1alwoxqbwp3t21g2qyt3yr8     worker-2   Ready    Active
36m5ii6eteslbleakl0zq4ddv     worker-3   Ready    Active
```

Gbr. 9 Node cluster swarm

3.) Implementasi NFS Volume Sharing

NFS *volume sharing* diterapkan untuk membuat suatu media penyimpanan persisten yang tersinkronisasi dan bisa diakses oleh banyak *node* dalam *cluster swarm.* Ketika *service* Moodle berjalan dan memiliki lebih dari satu replika

yang berada pada *node* yang berbeda, maka data *(moodlecode & moodledata)* diletakkan dalam direktori sharing NFS yang pada lingkungan docker disebut dengan *volume.*

Tabel 4. Instances NFS volume sharing

Instances	IP Address	Peran
NFS-Server	192.168.56.14	NFS Server
Swarm Worker2	192.168.56.12	NFS Client
Swarm Worker3	192.168.56.13	NFS Client

Konfigurasi NFS dilakukan dari sisi *nfs server* dan *nfs client*, mulai dari instalasi kerner nfs server, nfs plugin, dan pembuatan serta pengaturan hak akses direktori *sharing.*

Tabel 5. Pembuatan dan pengaturan hak akses direktori sharing

vagrant@nfs-server
sudo mkdir -p /var/volume_data
sudo chown -R nobody:nogroup /var/volume_data

Direktori *sharing* yang digunakan untuk menyimpan *volume* berlokasi di */var/volume_data,* kepemilikan direktori dibuat menjadi *nobody:nogroup* untuk memastikan pengaksesan direktori tanpa melaluiautentikasi kusus.

Tabel 6. Pemberian hak akses direktori sharing kepada nfs client

/etc/exports
/var/volume_data 192.168.56.12(rw,sync,no_root_squash,no_subtree_check) /var/volume_data 192.168.56.13(rw,sync,no_root_squash,no_subtree_check)

Pemberian hak akses terhadap direktori dilakukan melalui */exports,* pada bagian ini dilakukan inisialisasi alamat IP *node worker* yang diberikan akses terhadap direktori *sharing. Command rw* didefinisikan untuk memastikan *client* menapatkan hak membaca dan menulis pada direktori, *sync* untuk sinkronisasi direktori kepada *node* dalam lingkungan nfs, *no_root_squah* memungkinkan *client* melakukan akses terhadap direktori root, *no_subtree_check* untuk memastikan *client* dapat mengakses direktori dibawah /.

Tabel 7. Konfigurasi penerimaan hak akses direktori dari sisi client

/etc/fstab
192.168.56.14:/var/volume_data /var/volume_data nfs4 defaults,user,exec 0 0

Dari sisi *client (node worker-2 & worker-3),* konfigurasi dilakukan pada */fstab,* dengan memberikan *command* IP address dari *nfs server* serta *command* tambahan lain untuk menerima pemberian akses dari *nfs server.*

```
vagrant@worker-2:~$ sudo docker-volume-netshare nfs
INFO[0000] Checking for the references of volumes in docker daemon.
INFO[0015] == docker-volume-netshare :: Version: 0.36 - Built: 2019-
INFO[0015] Starting NFS Version 4 :: options: ''
```

Gbr. 10 NFS plugin pada node worker 2

```
vagrant@worker-3:~$ sudo docker-volume-netshare nfs
INFO[0000] Checking for the references of volumes in docker daemon.
INFO[0006] == docker-volume-netshare :: Version: 0.36 - Built: 2019
INFO[0006] Starting NFS Version 4 :: options: ''
```

Gbr. 11 NFS plugin pada node worker 3

Aktivasi *nfs plugin* dilakukan pada *node worker-2 dan worker-3* untuk memastikan *container* Moodle yang berjalan pada tersebut bisa membuat *volume* pada direktori *nfs sharing.*

4.) Implementasi Service

Setelah lingkungan server selesai dikonfigurasi, langkah selanjutnya adalah membuat layanan, dalam hal ini adalah *deployment* Moodle. Beberapa aplikasi yang dibutuhkan diantaranya Moodle, mariadb dan traefik. Pembuatan *service* didefinisikan dalam *stackfile* yang merupakan file dengan ekstensi .yml, sebuah file yang mempermudah kita untuk melakukan *deployment* banyak *service* sekaligus.

```
traefik:
  image: "traefik:v2.4"
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
  ports:
    - "80:8181"
    - "8080:8080"
  deploy:
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.api.service=api@internal"
    placement:
      constraints:
        - node.role == manager
  command:
    - --log.level=DEBUG
    - --api.insecure=true
    - --providers.docker=true
    - --providers.docker.exposedbydefault=false
    - --entrypoints.web.address=:8181
    - --api.dashboard=true
    - --api.debug=true
    - --providers.docker=true
    - --providers.docker.swarmMode=true
    - --providers.docker.exposedbydefault=false
    - --providers.docker.network=traefik-proxy
  networks:
    - traefik-proxy
```

Gbr. 12 Source code stackfile traefik

Image traefik menggunakan versi 2.4 yang di *pull* dari *docker registry*, dan secara *default* traefik akan menyimpan datanya di */var/run/docker.sock.* Traefik ditempatkan pada *node manager* dengan port yang digunakan yang pertama adalah 80:8181 dimana 80 merupakan port host dan 8181 merupakan port container, port 80:8181 ini digunakan traefik untuk fungsinya sebagai *edge router* yang nantinya IP dari manager bisa digunakan untuk mengakses halaman Moodle. Port traefik yang kedua adalah 8080:8080 yang digunakan untuk mengakses traefik dashboard. Label *traefic.enable=true* digunakan untuk memastikan fungsi traefik aktif, dan label

traefik.http.routers.api.service=api@internal untuk memanfaatkan api traefik sendiri sebagai edge router.

```
mariadb:
  image: bitnami/mariadb:latest
  environment:
   - BITNAMI_DEBUG=true
   - MARIADB_EXTRA_FLAGS= --max_allowed_packet=256M
   - MARIADB_USER=bn_moodle
   - MARIADB_DATABASE=bitnami_moodle
   - MARIADB_PASSWORD=bitnami
   - ALLOW_EMPTY_PASSWORD=yes
   - NAMI_LOG_LEVEL=trace8
  volumes:
    - "mariadb_data_vol:/bitnami/mariadb"
  deploy:
    mode: replicated
    replicas:  1
    placement:
      constraints:
        - node.role == worker
        - node.labels.worker==mariadb
  networks:
   - database
```

Gbr. 13 Source code stackfile mariadb

Implementasi *service* mariadb sebagai database server untuk Moodle. Image mariadb dari bitnami dengan tag *latest* (versi terbaru) yang di *pull* dari *docker registry*. Data mariadb disimpan dalam *disk* lokal dengan volume *mariadb_data_vol* yang berasosiasi dengan */bitnami/mariadb*. Mariadb ditempatkan pada *node worker-1*. *Environment variable* yang didefinisikan untuk menyimpan variabel konfigurasi mariadb mulai dari informasi nama *database, username,* dan *password.*

```
moodle:
  image: tegarshndrn/moodle:port-8181
  environment:
   - PHP_UPLOAD_MAX_FILESIZE=2048M
   - PHP_POST_MAX_SIZE=1024M
   #- MOODLE_SKIP_BOOTSTRAP=yes
   #- MOODLE_SKIP_INSTALL=yes
   - BITNAMI_DEBUG=true
   - MOODLE_DATABASE_HOST=mariadb
   - MOODLE_DATABASE_PORT_NUMBER=3306
   - MOODLE_DATABASE_USER=bn_moodle
   - MOODLE_DATABASE_NAME=bitnami_moodle
   - MOODLE_DATABASE_PASSWORD=bitnami
   - ALLOW_EMPTY_PASSWORD=yes
   - MOODLE_USERNAME=tegarshndrn16
   - MOODLE_PASSWORD=AdmIntegarshndrn16
   - MOODLE_EMAIL=tegarshndrn16@gmail.com
   - MOODLE_SITE_NAME='Moodle Cluster Docker Swarm'
  volumes:
   - moodle_data_vol:/bitnami/moodle
   - moodledata_data_vol:/bitnami/moodledata
  depends_on:
   - mariadb
  deploy:
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.moodle.rule=Host(`192.168.56.10`)"
      - "traefik.http.routers.moodle.entrypoints=web"
      - "traefik.http.services.moodle.loadbalancer.server.port=8181"
    mode: replicated
    replicas:  1
    placement:
      max_replicas_per_node: 5
      constraints:
        - node.role == worker
        - node.labels.worker==moodle
  networks:
   - database
   - traefik-proxy
```

Gbr. 14 Source code stackfile moodle

Implementasi *service moodle* pada penelitian ini diletakkan pada *node worker 2* dan *node worker 3* dan menggunakan image Moodle milik penulis yang dipublikasi di *docker registry* dengan tag *tegarshndrn/moodle:port-8181*, *image moodle* ini dibuat untuk menyesuaikan penggunaan *port*, karena *port* 80 dipakai untuk traefik sehingga Moodle yang digunakan adalah dengan *port* 8181, konfigurasi *port* ini didefinisikan di *dockerfile.* Pada *deployment* pertama, Moodle didefinisikan hanya dengan satu replika, setelah *service* berjalan dilakukan *scaling* menyesuaikan kebutuhan jumlah replika. *Environment variable* yang didefinisikan untuk menyimpan variabel konfigurasi Moodle mulai dari informasi *database, username*, dan *password*. Dalam *service* Moodle ini juga ditambahkan label untuk konektivitas dengan traefik dengan *host* ip 192.168.56.10 yang merupakan ip *manager* dimana *container* traefik berjalan. Setelah *container* Moodle dijalankan, secara otomatis akan melakukan *tracking* terhadap dua folder penting yaitu *moodle code* dan *moodle data,* sehingga dua folder tersebut disimpan dalam dua *volume* yang berbeda yaitu *moodle_data_vol* dan *moodledata_data_vol.* Moodle berkaitan langsung dengan mariadb dan traefik, sehingga menggunakan dua *network* yang berafiliasi dengan dua *service* tersebut, yaitu *network database* dan *network traefik-proxy*.

```
networks:
  database:
    driver: overlay
    external: true
  traefik-proxy:
    driver: overlay
    external: true
```

Gbr. 15 Source code stackfile konfigurasi network

Saat menginstall *docker engine*, secara otomatis dalam docker juga terinstall *network* dengan tipe *bridge*. Untuk *swarm cluster* yang menghubungkan *container* antar *node*, maka *driver network* yang digunakan adalah dengan tipe *overlay*. Pada penelitian ini menggunakan dua *network* yang diinisialisasi dengan nama *database* dan *traefik-proxy*, keduanya merupakan *overlay network*. *Network database* untuk komunikasi mariadb dengan Moodle dan *network traefik-proxy* untuk komunikasi traefik dan Moodle.

```
volumes:
  mariadb_data_vol:
  moodle_data_vol:
    driver: nfs
    driver_opts:
      share: 192.168.56.14:/var/volume_data/moodle_data
  moodledata_data_vol:
    driver: nfs
    driver_opts:
      share: 192.168.56.14:/var/volume_data/moodledata_data
```

Gbr. 16 Source code stackfile konfigurasi volume

Pada penelitian ini, *docker volume* yang disimpan dalam *disk* lokal sebanyak tiga *volume*, yaitu *mariadb_data_vol* untuk mariadb, *moodle_data_vol* untuk *moodle code* dan *moodledata_data_vol* untuk *moodle data*. Pendefinisian *mariadb_data_vol* lebih sederhana karena berada di satu *node* saja. Berbeda dengan *moodle_data_vol* dan *moodledata_data_vol,* karena keduanya merupakan *swarm volume* yang merupakan volume dengan basis direktori NFS server dengan alamat 192.168.56.14:*/var/volume_data*, hal ini diterapkan karena Moodle memiliki replika lebih dari satu dan berada pada dua *node* yang berbeda, sehingga tipe *volume* yang digunakan harus mampu melakukan sinkronisasi data agar bisa diakses dari beberapa *node* yang berbeda.

5.) Implementasi Load Balancing

```
command:
  - --log.level=DEBUG
  - --api.insecure=true
  - --providers.docker=true
  - --providers.docker.exposedbydefault=false
  - --entrypoints.web.address=:8181
  - --api.dashboard=true
  - --api.debug=true
  - --providers.docker=true
  - --providers.docker.swarmMode=true
  - --providers.docker.exposedbydefault=false
  - --providers.docker.network=traefik-proxy

labels:
  - "traefik.enable=true"
  - "traefik.http.routers.moodle.rule=Host(`192.168.56.10`)"
  - "traefik.http.routers.moodle.entrypoints=web"
  - "traefik.http.services.moodle.loadbalancer.server.port=8181"
```

Gbr. 17 Source code stackfile konfigurasi load balancing

Command traefik memuat konfigurasi traefik sebagai *load balancer. --api.insecure=true* traefik akan mendengarkan pada port 8080 secara default untuk permintaan API, *--providers.docker=true* untuk mengaktifkan provider docker, *--entrypoints.web.address=:8181* mengartikan bahwa traefik akan mendengarkan permintaan yang masuk pada port 8181, *--providers.docker.exposedbydefault=false* mencegah ekspos kontainer kecuali yang terdata secara eksplisit.

Dari sisi Moodle, ditambahkan label untuk konektivitas dengan traefik sebagai *load balancer.* - *"traefik.enable=true"* untuk memberitahu traefik secara eksplisit agar mengekspos container Moodle, - *"traefik.http.routers.moodle.rule=Host(`192.168.56.10`)"* alamat IP atau domain yang akan diberikan respon, - *"traefik.http.routers.moodle.entrypoints=web"* menerima permintaan dari *entry point* yang didefinisikan dengan nama web, - *"traefik.http.services.moodle.loadbalancer. server.port=8181"* menerima *service* dari port 8181 dalam *load balancing*.
Setelah konfigurasi didefinisikan, saat *service* dijalankan traefik akan mengatur pembagian tugas kepada *instance* dalam *cluster* secara otomatis.

6.) Deployment System

Di lingkungan docker, *deployment service* yang dirancang dalam format *compose* maupun *stackfile* dilakukan menggunakan perintah dasar yang ditentukan oleh docker.

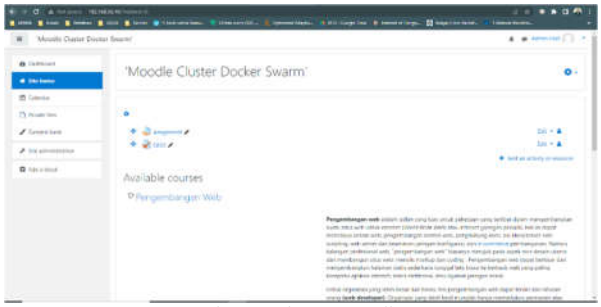
Tabel 14. Command deployment system

vagrant@manager
Docker stack deploy –compose-file=mdl-stackfile-multi.yml moodle

Untuk memastikan *deployment* berhasil dilakukan, dilakukan pengecekan terhadap service yang berjalan di node, dengan perintah *docker service ls.*

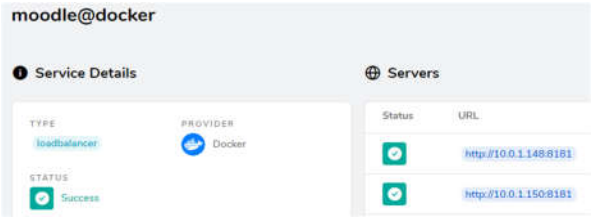
Gbr.10 Docker service pada awal deployment

Mengakses halaman Moodle, menggunakan ip 192.168.56.14, ip tersebut merupakan ip milik manager yang digunakan oleh traefik sebagai *edge router* dan *load balancer* agar *entry point* menuju Moodle terpusat di satu alamat tersebut.



Gbr.11 Halaman Moodle

mengakses dashboard, kita bisa mengunjungi alamat 192.168.56.10:8080, port tersebut dialokasikan untuk traefik dashboard sesuai dengan yang dituliskan pada stackfile. Pada traefik dashboard ada banyak informasi yang ditampilkan dengan *user interface* yang menarik dan mudah dipahami. Dengan tiga replika Moodle, pada dashboard traefik ditampilkan *load balancer* dengan jumlah tiga dengan ip yang berbeda tiap replika.



Gbr.12 Dashboard Traefik

B. Hasil Pengujian Kinerja, Pengujain Scaling, dan Pengujian Fungsional

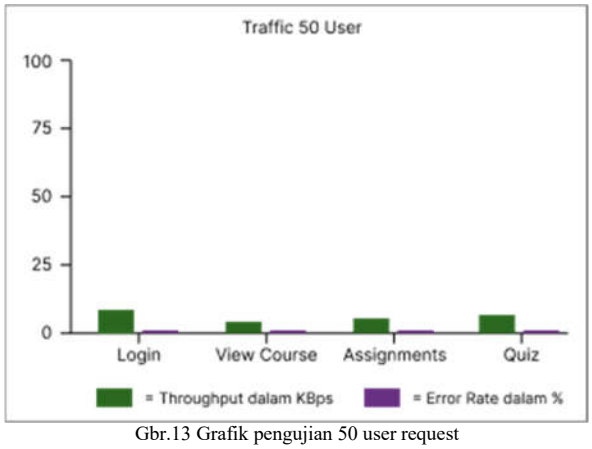
Pengujian dilakukan sebagai upaya untuk mengetahui kinerja moodle, serta mekanisme *scaling* dan mekanisme *failover* pada *swarm cluster* Berikut hasil pengujian menggunakan apache jmeter:

1) Pengujian Kinerja Server Moodle tanpa Load Balancing

Pengujian ini dilakukan untuk mengetahui kinerja dari *single server* Moodle dalam merespon aktivitas user yang meliputi *login, view course, assignment,* dan *quiz* dengan parameter yang digunakan dalam pengujian ini yaitu *throughput* dan *error rate.*

Tabel 15. Hasil pengujian 50 user request

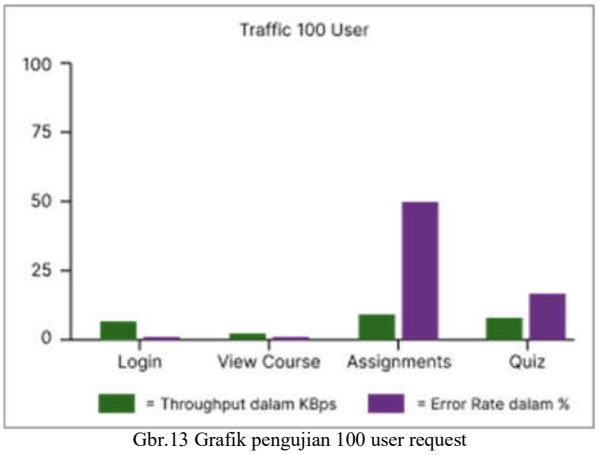
Aktivitas User	Throughput (KBps)	Error Rate (%)
Login	9.4	0
View Course	4.4	0
Assignments	5.7	0
Quiz	6.4	0



Gbr.13 Grafik pengujian 50 user request

Tabel 16. Hasil pengujian 100 user request

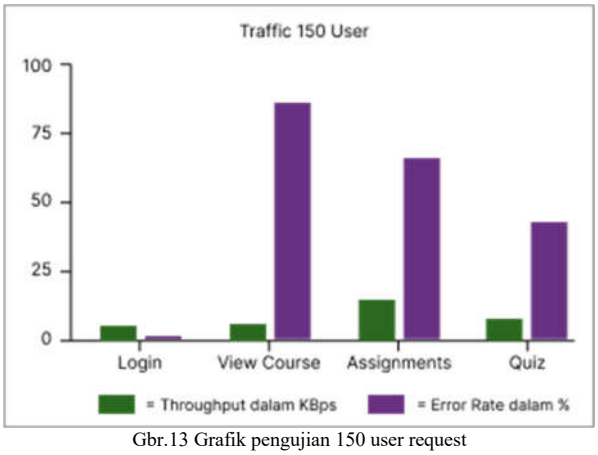
Aktivitas User	Throughput (KBps)	Error Rate (%)
Login	7.0	0
View Course	2.1	0
Assignments	11.3	50.00
Quiz	9.1	19.00



Gbr.13 Grafik pengujian 100 user request

Tabel 17. Hasil pengujian 150 user request

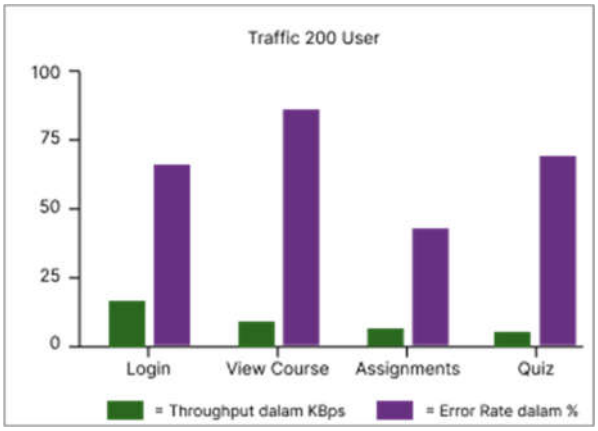
Aktivitas User	Throughput (KBps)	Error Rate (%)
Login	6.2	0
View Course	8.4	86.00
Assignments	14.3	66.67
Quiz	9.5	46.67



Gbr.13 Grafik pengujian 150 user request

Tabel 18. Hasil pengujian 200 user request

Aktivitas User	Throughput (KBps)	Error Rate (%)
Login	18.3	60.00
View Course	9.3	87.50
Assignments	7.4	40.00
Quiz	7.1	60.00



Gbr.13 Grafik pengujian 200 user request

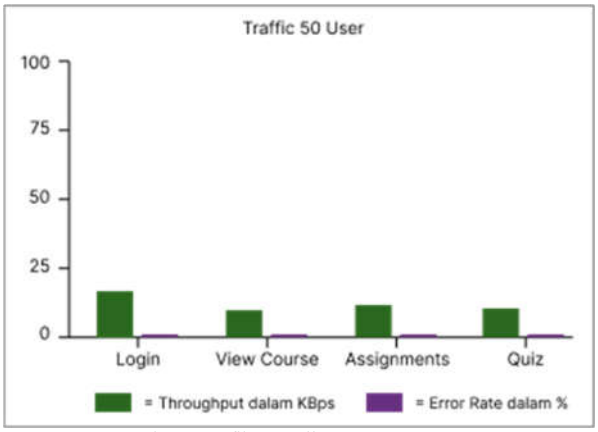
Berdasarkan hasil pengujian kinerja Moodle dengan *single server*, terlihat bahwa terdapat penurunan *throughput* seiring dengan meningkatnya *request user*. Kondisi tersebut menunjukkan ketidakstabilan server dan cenderung memiliki kinerja yang kurang baik ketika *request user* diatas 50. Mengacu pada parameter *load server Moodle* pada situs *lms.onnocenter.id* yang menyebutkan bahwa pengukuran jumlah maksimal *request user* dalam satu waktu adalah dengan perhitungan kapasitas RAM dalam satuan *Giga Byte* (GM) dikalikan 50. Pada pengujian *single server* Moodle ini, server yang menjalankan Moodle memiliki RAM 1 GB, dengan perhitungan tersebut idealnya *server cluster* pada penelitian ini mampu menangani 50 *request user* dalam satu waktu, sedangkan hasil yang didapat dalam pengujian hingga 100, 150, dan 200 *request user,* respon server mengalami penurunan karena jumlah tersebut diluar kapasitas server.

2) Pengujian Kinerja Server Moodle dengan Load Balancing

Pengujian ini dilakukan untuk mengetahui kinerja dari *load balancing server* Moodle dalam merespon aktivitas user yang meliputi *login, view course, assignment,* dan *quiz* dengan parameter yang digunakan dalam pengujian ini yaitu *throughput* dan *error rate.*

Tabel 19. Hasil pengujian 50 user request

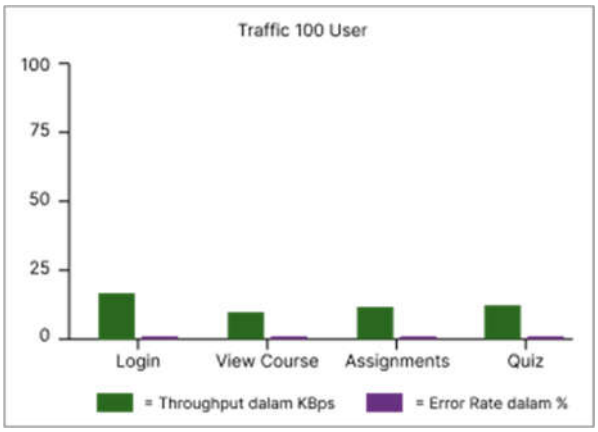
Aktivitas User	Throughput (KBps)	Error Rate (%)
Login	17.8	0
View Course	10.4	0
Assignments	13.5	0
Quiz	13.3	0



Gbr.13 Grafik pengujian 50 user request

Tabel 20. Hasil pengujian 100 user request

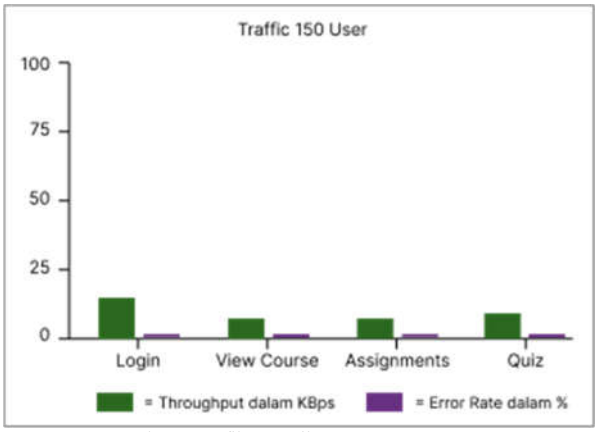
Aktivitas User	Throughput (KBps)	Error Rate (%)
Login	17.8	0
View Course	8.0	0
Assignments	10.4	0
Quiz	10.9	0



Gbr.13 Grafik pengujian 100 user request

Tabel 21. Hasil pengujian 150 user request

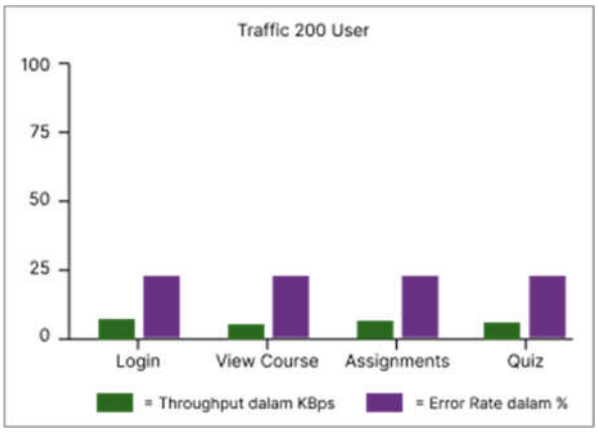
Aktivitas User	Throughput (KBps)	Error Rate (%)
Login	14.4	0
View Course	7.6	0
Assignments	7.2	0
Quiz	9.0	0



Gbr.13 Grafik pengujian 150 user request

Tabel 22. Hasil pengujian 200 user request

Aktivitas User	Throughput (KBps)	Error Rate (%)
Login	8.5	24.50
View Course	5.6	24.50
Assignments	7.7	24.50
Quiz	7.6	24.50



Gbr.13 Grafik pengujian 200 user request

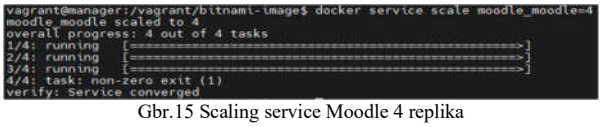
Berdasarkan hasil pengujian kinerja server Moodle dengan *container load balancing,* terlihat bahwa terdapat penurunan *throughput* seiring dengan meningkatnya *request user*. Namun kondisi tersebut masih dalam batas normal dan cenderung memiliki kinerja yang bagus. Mengacu pada parameter *load server Moodle* pada situs *lms.onnocenter.id* yang menyebutkan bahwa pengukuran jumlah maksimal *request user* dalam satu waktu adalah dengan perhitungan kapasitas RAM dalam satuan *Giga Byte* (GM) dikalikan 50. Pada penelitian ini, server yang menjalankan Moodle sebanyak 2 server dengan masing-masing memiliki RAM 1 GB, dengan perhitungan tersebut idealnya *server cluster* pada penelitian ini mampu menangani 100 *request user* dalam satu waktu, sedangkan hasil yang didapat dalam pengujian hingga 150 *request user*, respon dari server masih normal tanpa adanya error. Kendala mulai didapati ketika *request user* berjumlah 200 dimana jumlah tersebut diluar kapasitas server.

3) Pengujian Scaling

Pengujian *scaling* dilakukan dengan prosedur *scaling* pada *docker swarm*, yaitu dengan menjalankan perintah untuk menyesuaikan jumlah replikasi. Jumlah *service* Moodle sebelum di replika berjumlah 1 dan berada pada node worker 3.

Gbr.13 Scaling service Moodle 2 replika

Gbr.14 Service Moodle setelah scaling 2 replika



Gbr.15 Scaling service Moodle 4 replika

Gbr.16 Service Moodle setelah scaling 4 replika

Sebelum pengujian, *container* Moodle berjumlah satu replika dan berada pada *node worker*-3, setelah dilakukan *scaling* sebanyak empat replika, sebanyak dua *container* Moodle secara otomatis oleh swarm ditempatkan pada *node worker*-3 dan dua container moodle ditempatkan pada *node worker*-2. Sistem tetap berjalan dengan baik pasca dilakukan *scaling*.

4) Pengujian Failover

Pengujian *failover* sistem dilakukan dalam rangka untuk mengetahui apakah sistem yang dibangun mengalami *error* atau berjalan dengan baik. Mekanisme pengujian fungsional dilakukan menggunakan konsep *failover,* dimana akan dilakukan pengujian dengan mematikan salah satu mesin atau node kemudian melakukan pengecekan apakah tugas dari node yang dimatikan bisa otomatis dilimpahkan ke node lain dan sistem tetap berjalan dengan normal. Pengujian ini sebagai upaya memastikan tingkat ketersediaan server pada *cluster swarm.* Pengujian dilakukan dengan cara mematikan salah satu host yang ada pada *swarm cluster.*

Gbr.17 Kondisi service moodle sebelum pengujian fungsional

Gbr.18 Kondisi service Moodle setelah pengujian fungsional

Dari hasil pengujian *failover* menunjukkan proses *failover* dari docker swarm dalam mengatasi *node worker*-3 yang di *shutdown*, secara otomatis *service* pada *node worker*-3 dialihkan ke *node worker*-2 yang berjalan normal. Secara keseluruhan pengujian fungsional berjalan dengan baik sesuai dengan mekanisme *failover* yang dimiliki oleh docker swarm.

IV. KESIMPULAN

Dari penelitian ini penulis berhasil menerapkan lingkungan server *docker swarm cluster* untuk *deployment* Moodle dengan menambahkan implementasi metode *container load balancing* untuk Moodle server. Setelah dilakukan pengujian, dihasilkan kesimpulan bahwa server mampu melakukan manajemen *load traffic* dari *user request* dengan baik dilihat dari parameter *throughput* dan *error rate* yang diperoleh pada tahap pengujian kinerja. NFS mampu menyediakan penyimpanan data persisten untuk volume *Moodle code* dan *Moodle data* yang dibutuhkan Moodle dalam melakukan *tracking* terhadap volume tersebut. Mekanisme *scaling* dan *failover* docker swarm berjalan dengan baik sehingga server memiliki ketersediaan tingkat tinggi

V. SARAN

Berdasarkan hasil dari penelitian ini, peneliti memberikan saran untuk penelitian serupa kedepannya. Penentuan algoritma *load balancing* bisa dilakukan untuk menyesuaikan dengan kebutuhan lingkungan server, serta mengacu pada tingkat penggunaan *resource*, *deployment* di lingkungan *cloud* menjadi pertimbangan yang cukup penting untuk meningkatkan aksesibilitas, dan *deployment* aplikasi dengan kompleksitas tinggi, penggunaan kubernetes sebagai orkestrator docker menjadi pertimbangan yang baik.

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. Terimakasih kepada diri sendiri karena dapat berkompromi untuk menggapai tujuan yang ingin dicapai.

REFERENSI

[1] Ahmad Aji Santoso, Adhitya Bhawiyuga & Reza Andria Siregar. "Evaluasi Kinerja Moodle *E-Learning* pada *Cluster Docker Swarm* di Amazon Web Services". Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, Vol 6. No. 3 2022.

[2] Andreas Frederius. "Implementasi Penyimpanan Data Persisten Pada *Docker Swarm* Menggunakan *Network File System* (NFS)". Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, Vol 3. No. 2 2018.

[3] Ayman Elsayed Khedr. "*Adapting Load Balancing Technique for Improving the Performance of e-Learning Educational Process"*. Journal of Computer, Vol. 12 No. 3 2017.

[4] Dimas Setiawan Afis. "*Load Balancing* Server Web Berdasarkan Jumlah Koneksi Klien Pada Docker Swarm". Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, Vol 3. No. 1 2018.

[5] M Fadlulloh Romadhon Bik & Asmunin.. "Implementasi Docker Untuk Pengelolaan Banyak Aplikasi Web (Studi Kasus: Jurusan Teknik Informatika Unesa)". Jurnal Manajemen Informatika, Vol. 7 No 2 2017.

[6] Mohamad Rexa Mei Bella, Mahendra Data, & Widhi Yahya. "Implementasi *Load Balancing* Server Web Berbasis Docker Swarm Berdasarkan Penggunaan Sumber Daya *Memory Host".* Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, Vol 3. No. 4 2019.

[7] Muhammad Fihri, Ridha Muldina Negara & Danu Dwi Sanjoyo. "Implementasi & Analisis Performansi Layanan Web Pada Platform Berbasis Docker". E-Proceeding of Engineering, Vol. 6 No. 2 2019.

[8] Mujiono Sadikin, Raka Yusuf & Arif Rifai D. "*Load Balancing Clustering on Moodle LMS to Overcome Performance Issue on E-Learning System".* Telekomunika, Vol. 17 No. 1 2019.

[9] Noer Ekafitri Sam & Reski Idrus. "Efektifitas Media ELearning Berbasis *Learning Management System* (LMS) di Era Pandemi Covid-19". Jurnal Ikraith-Humaniora, Vol. 5 No. 3 2021.

[10] (2020) Onno Widodo Purbo. Moodle: *Load Server Moodle*. lms.onnocenter.or.id.

[11] 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 2021.

[12] Tanjung P Kusuma, Rendy Munadi, & Danu Dwi Sanjoyo. “Implementasi dan Analisis Computer Clustering System dengan Menggunakan Virtualisasi Docker”. e-Proceeding of Engineering, Vol.4 No.3 2017.