IMPELEMENTASI LOAD BALANCING DENGAN METODE LEAST CONNECTION PADA ARSITEKTUR SERVER LMS MOODLE BERBASIS CLOUD DAN DOCKER

Dody Prasetyo[1)] **I Putu Satwika**[2)] **I Made Artana**[3)]
Program Studi Teknik Informatika[1)2)3)]
Sekolah Tinggi Manajemen Informatika dan Komputer (STMIK) Primakara, Denpasar, Bali[1) 2) 3)]
prasetyodody17@gmail.com[1)]satwika@primakara.ac.id[2)]madeartana88@gmail.com[3)]

ABSTRACT

Internet is the easiest thing to be enjoyed by anyone today. If we want to share a photo or video on the internet, we just need to press a few buttons on a website and quickly our photo or video will spread out and can be seen by anyone. But have we ever thought how can we upload photos or videos, when millions or even billions of other users are browsing the site or uploading photos too. This research will examine these kinds of topic with a case study from one of the projects by STMIK Primakara Campus, namely Ruang Kuliah. Ruang Kuliah are predicted to be accessed by many visitors and lecturers from various campuses and universities in Indonesia. Departing from these problems, a correct architectural design is needed and it expected able to overcome conditions when site traffic is on high demand. Research was carried out and finally found a technique called ***Load Balancing****. However, after testing, it turned out that Load Balancing on this case study did not give satisfactory results. Because when hit by 13000 requests, the response time given by the website reaches* ***28504ms*** *or* ***28 seconds****.*
Keywords: *load balancing, least connection, moodle LMS*

ABSTRAK

Internet merupakan hal yang paling mudah untuk dinikmati oleh siapa saja saat ini. Jika kita ingin menyebarkan suatu foto ataupun video ke internet, kita hanya cukup menekan beberapa tombol pada suatu situs jejaring sosial dan dengan cepat foto atau video kita tersebar dan dapat di lihat oleh siapapun saja. Tapi pernahkah kita berfikir bagaimana bisa kita melakukan upload foto ataupun video, disaat berjuta-juta bahkan milyaran pengguna situs yang lain sedang menjelajahi situs atau melakukan upload juga. Penelitian kali ini akan mengangkat topik permasalahan tersebut dengan studi kasus permasalahan dari salah satu proyek website dari Kampus STMIK Primakara yaitu Ruang Kuliah. Ruang Kuliah di prediksikan akan di akses oleh banyak pengunjung maupun dosen dari berbagai kampus dan universitas yang ada di Indonesia. Berangkat dari permasalahan tersebut maka dibutuhkan rancangan arsitektur yang benar dan mampu mengatasi kondisi ketika pengunjung situs sedang ramai yang menjadi studi kasus dalam penelitian ini. Penelitian pun dilakukan dan akhirnya ditemukanlah teknik yang dinamakan ***Load Balancing***. Namun setelah dilakukannya pengujian, dengan *Load Balancing* ternyata pada studi kasus kali ini tidak memberikan hasil yang memuaskan. Dikarenakan ketika di hit hingga 13000 requests, lama waktu respon yang diberikan website mencapai **28504ms** atau **28 detik**.
Kata Kunci: *load balancing, least connection,* moodle LMS

PENDAHULUAN

Dalam rangka menyiapkan mahasiswa dalam menghadapi perubahan sosial, budaya dan dunia kerja serta kemajuan teknologi yang pesat, baru-baru ini Direktorat Jenderal Pendidikan Tinggi dan Kebudayaan Indonesia meluncurkan kebijakan baru yang bernama Kampus Merdeka. Kampus Merdeka merupakan kebijakan Menteri Pendidikan dan Kebudayaan yang bertujuan mendorong mahasiswa untuk menguasai berbagai keilmuwan yang berguna untuk memasuki dunia kerja[1]. Melalui Kampus Merdeka, mahasiswa memiliki kesempatan untuk menempuh pembelajaran di luar program studi selama satu semester atau setara dengan 20 SKS pada perguruan tinggi yang sama. Dan juga memiliki kesempatan untuk menempuh pembelajaran pada program studi yang berbeda di perguruan tinggi yang berbeda selama paling lambat dua semester atau setara 40 SKS.

Dalam mendukung kebijakan tersebut, STMIK Primakara memiliki inovasi dengan menciptakan suatu *platform* kursus *online* berbasis *website* yang bernama Ruang Kuliah. Ruang Kuliah adalah *platform* kursus *online* dimana setiap kursus nya memiliki poin yang nantinya dapat dijadikan pengganti SKS dalam mata kuliah formal dalam rangka merealisasikan kebijakan Kampus Merdeka dari Kementrian Pendidikan dan Kebudayaan. *Platform* Ruang Kuliah dibangun menggunakan salah satu *Learning Management System* atau biasa disingkat *LMS* yang diciptakan secara *open source* bernama Moodle. Moodle merupakan *LMS* yang paling populer dan paling banyak digunakan di dunia[2]. Ruang Kuliah di prediksikan akan di akses oleh banyak pengunjung maupun dosen dari berbagai kampus dan universitas yang ada di Indonesia. Sehingga dibutuhkan arsitektur *server* yang dapat mampu bertahan jika sewaktu-waktu terjadi lonjakan jumlah pengunjung yang mengakses dan berinteraksi pada *website*.

Tentu dalam mengatasi permasalahan tersebut biasanya kita dengan mudah mengambil keputusan untuk melakukan peningkatan spesifikasi atau kapasitas dari *server* yang digunakan, atau bisa disebut melakukan *Vertical Scaling*[3]. Namun jika kita lihat dari sisi finansial, melakukan *Vertical Scaling* akan memakan biaya yang sangat tinggi dikarenakan semakin besarnya kapasitas yang dibutuhkan semakin besar juga biaya yang harus kita bayarkan. Selain itu kita harus membayar biaya yang bersifat tetap atau *Fixed* walaupun tingkat permintaan pelayanan ke *server* sedang dalam kondisi tidak tinggi. Oleh karena itu ditemukanlah teknik yang dinamakan *Horizontal Scaling*. *Horizontal Scaling* merupakan teknik yang dapat mendistribusikan beban kerja, jaringan, atau lalu lintas aplikasi secara merata di beberapa *server*[3].

Konsep dari *Horizontal Scaling* adalah melakukan *scaling* dengan penambahan unit *server* dengan kapasitas yang sama persis dengan *server* utama, kemudian koneksi yang datang akan di di bagi ke beberapa unit *server* yang ada. Ketika *traffic* dirasa sudah sepi kita dapat mengurangi jumlah unit *server* yang aktif. Sehingga biaya yang dikeluarkan cukup murah karena hanya membayar *server* berkapasitas kecil. *Load Balancing* merupakan salah satu metode *scaling* yang melakukan *scaling* secara *Horizontal*. Perangkat yang melakukan *Load Balancing* disebut *Load Balancer*.

Layanan arsitektur yang saat ini dapat melakukan *Load Balancing* adalah arsitektur server dengan model komputasi awan yang biasa disebut *Cloud Computing*. Pada layanan *Cloud Computing* kita dapat dengan bebas menambahkan ataupun mengurangi *server* tanpa memikirkan jumlah *server* yang tersedia. Pada layanan ini kita cukup membayar sesuai dengan penggunaan, kapasitas dan jumlah *cloud server* yang di pesan. Saat ini terdapat banyak sekali layanan komputasi awan yang dapat di sewa dengan mudah di internet salah satunya adalah Digital Ocean, yang akan dipergunakan menjadi *server* yang akan digunakan pada studi kasus kali ini. Digital Ocean juga menyediakan paket *Load Balancer* yang dapat dipasang pada arsitektur kita, namun untuk menekan biaya *Development* dari Ruang Kuliah lebih jauh lagi, pada penelitian ini akan mencoba menggunakan salah satu perangkat lunak yang gratis yaitu NGINX. NGINX adalah perangkat lunak *web server* yang memiliki fitur *Load Balancer* didalamnya. Kita dapat menggunakan tiga algoritma dalam

proses pendistribusian *traffic* secara gratis pada *Load Balancer* ini, yaitu *Roundrobin, IP Hash, Least Connection*[9]. Pada penelitian sebelumnya berjudul *"Implementasi Load Balancing Pada Web Server Menggunakan Nginx"* oleh F. Apriliansyah dkk, menemukan bahwa algoritma terbaik dari ketiga algoritma tersebut jatuh pada *Least Connection*. Berangkat dari hasil tersebut maka pada studi kasus ini peneliti akan mencoba menguji menggunakan algoritma *Least Connection* yang dikemas didalam *Container Docker*.

TINJAUAN PUSTAKA

Cloud Computing

Cloud Computing merupakan hasil kombinasi dari sistem terdistribusi, komputasi grid, dan virtualisasi dengan beberapa konsep baru. Konsep baru ini meningkatkan kinerja dan layanan yang diberikan kepada pengguna dengan menyediakan sistem komputasi yang berbeda dari teknologi-teknologi sebelumnya[2].

Cloud Computing Delivery Service

Cloud computing delivery service adalah produk layanan, solusi, dan model bisnis yang memenuhi kebutuhan pengguna melalui Internet[2] *Cloud Computing Delivery Service* dapat terbagi menjadi 3 jenis[3], yaitu *Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS).*

Virtualisasi

Virtualisasi merupakan representasi logis dari komputer dalam perangkat lunak. Dengan memisahkan *hardware* dari sistem operasi, virtualisasi menyediakan fleksibilitas operasional yang lebih meningkatkan tingkat pemanfaatan perangkat keras fisik yang mendasarinya[6]. Virtualisasi terbagi menjadi virtualisasi perangkat keras dan virtualisasi data. Dengan adanya virtualisasi akan memudahkan pengguna untuk dapat menjalankan sistem operasi yang berbeda dari semua mesin virtual pada satu mesin fisik. Ketika terjadi kesalahan maupun kekeliruan dalam penginstalan program, tidak akan mempengaruhi mesin virtual yang lainnya. Pengguna dapat dengan bebas melakukan eksperimen tanpa perlu khawatir dengan hilangnya maupun rusaknya program serta data bawaan yang ada.

Container

Container adalah adalah unit software yang mengemas semua *code* dengan dependensinya sehingga aplikasi dapat berjalan dengan cepat dari satu lingkungan komputasi ke lingkungan komputasi yang lain[7]. Dengan kata lain *container* dapat diasumsikan sebagai sebuah wadah yang dapat menampung banyak data maupun aplikasi dengan tujuan menjadi lebih ringkas untuk dijalankan pada sistem. Perbedaan *infrastructure* virtualisasi dengan *container* adalah dimana *container* hanya membutuhkan *library* serta aplikasi yang digunakan tanpa membutuhkan keseluruhan perangkat keras, *kernel*, dan *OS*. Berbeda dengan virtualisasi yang membutuhkan keseluruhan sistem.

Scalability

Bisnis terus bertumbuh seiring berjalannya waktu, jumlah pelanggan dan penggunaan sumber daya semakin lama semakin meningkat. Jika perusahaan tidak siap meningkatkan sistem yang di gunakannya dalam menangani pertumbuhan tambahan penggunaan sumber daya, maka sistem yang dibuat oleh perusahaan tersebut tidak dapat mengimbangi pertumbuhan yang telah terjadi pada perusahaan.

Untuk tetap kompetitif, perusahaan harus siap untuk memenuhi kebutuhan dan keinginan pelanggan. Sistem harus ditingkatkan pada satu atau lebih level, untuk memenuhi kebutuhan bisnis. Jika tidak, reputasi bisnis akan terganggu yang mengakibatkan hubungan pelanggan yang buruk. Lalu muncul pertanyaan, bagaimana cara memecahkan permasalahan tersebut? Jawabannya adalah dengan melakukan *Scalability*. Terdapat dua teknik dalam melakukan *Scalability*[9], yaitu *Vertical Scaling* dan *Horizontal Scaling*.

Load Balancing

Load balancing merupakan teknik yang dapat mendistribusikan beban kerja, jaringan, atau lalu lintas aplikasi merata di beberapa *server*[9]. Fungsi *load balancing* sama dengan polisi lalu lintas, dan tugasnya adalah mencegah kemacetan lalu lintas dan kecelakaan

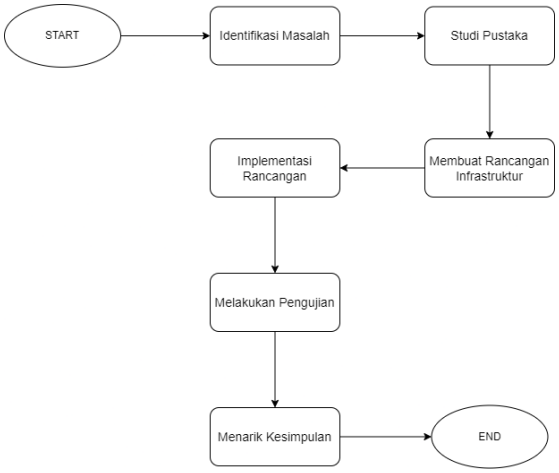
lalu lintas yang tidak terduga. Load Balancer harus dapat menjamin kelancaran arus lalu lintas jaringan dan sekaligus memberikan rasa aman dalam sistem kerja jaringan yang kompleks. Dalam *Load Balancing*, terdapat 5 metode yang dapat dipilih dalam melakukan pengaturan *load balancing*[9], yaitu *Roundrobin, Least Connection, Least Response Time, Least Bandwidth, IP Hash.*

Docker

Docker adalah *tools* berbasis *open source* yang dapat secara otomatis menyebarkan aplikasi kedalam suatu wadah yang bernama *container*. Docker adalah *platform* yang dibangun di atas teknologi *container*[8]. *Container Image* adalah paket perangkat lunak ringan independen yang berisi segala hal yang diperlukan suatu os untuk berjalan: kode, *runtime*, alat sistem, pustaka sistem, dan pengaturan. Dibandingkan dengan virtualisasi, perbedaan paling signifikan antara container adalah memiliki *file* yang lebih kecil karena tidak memerlukan sistem operasi yang lengkap.

METODE PENELITIAN

Kerangka Berpikir



Gambar 1. Kerangka Berpikir

Metode Penelitian

Untuk mengetahui apakah rancangan yang dibuat pada penelitian ini benar-benar menyelesaikan permasalahan dari studi kasus, pada penelitian ini akan dilakukan perbandingan antara rancangan yang telah melakukan *Load Balancing* dengan yang tidak melakukan *Load Balancing*. Mengacu pada penelitian serupa yang telah di lakukan sebelumnya, alat pengujian yang digunakan pada penelitian ini adalah Apache JMeter. Untuk jumlah *request* pengujian yang digunakan pada penelitian ini, berdasarkan hasil *research* pada situs *Development* Moodle yang telah dilakukan, Moodle secara resmi telah menyiapkan Apache JMeter *Scripts* yang dapat digunakan untuk pengujian dengan beberapa kategori jumlah *user*. Berikut adalah kategori jumlah *user* yang dipergunakan pada penelitian ini.

No	Aksi	Jumlah User
1	Akses Halaman Depan	30
2	Akses Halaman *Login*	30
3	*POST Login*	30
4	Akses Halaman Dashboard	30
5	Akses Halaman Kursus	30
6	Akses Halaman *Activity*	30
7	Akses Halaman Kursus Lagi	30
8	Akses Halaman Diskusi	30
9	Akses Form Reply Diskusi	30
10	POST Form Reply Diskusi	30
11	Akses Halaman Kursus Lagi	30
12	Akses Halaman Partisipan Kursus	30
13	POST Logout	30

Tabel 1. 390 *User Request*

Pada kategori ini *server* Moodle akan diuji dengan total *dummy user* sebanyak 30 *User* dengan total 13 aksi. Dimana masing-masing aksi akan diakses oleh 30 *User* sehingga total request yang di jalankan pada kategori ini adalah **390 *Requests***.

No	Aksi	Jumlah User
1	Akses Halaman Depan	100
2	Akses Halaman *Login*	100
3	*POST Login*	100
4	Akses Halaman Dashboard	100
5	Akses Halaman Kursus	100
6	Akses Halaman *Activity*	100
7	Akses Halaman Kursus Lagi	100
8	Akses Halaman Diskusi	100
9	Akses Form Reply Diskusi	100
10	POST Form Reply Diskusi	100
11	Akses Halaman Kursus Lagi	100
12	Akses Halaman Partisipan Kursus	100
13	POST Logout	100

Tabel 2. 1300 *User Request*

Masih sama dengan kategori sebelumnya, pada kategori ini juga akan diuji dengan aksi sebanyak 13 aksi. Namun yang berbeda pada kategori ini adalah jumlah *user* yang melakukan permintaan aksi tersebut, yaitu sebanyak 100 *user*. Sehingga terdapat 13 aksi dengan masing-masing 100 *user* yang melakukan permintaan aksi tersebut pada *server*, sehingga total request yang berjalan pada kategori ini adalah **1300 *Requests***.

No	Aksi	Jumlah User
1	Akses Halaman Depan	1000
2	Akses Halaman *Login*	1000
3	*POST Login*	1000
4	Akses Halaman Dashboard	1000
5	Akses Halaman Kursus	1000
6	Akses Halaman *Activity*	1000
7	Akses Halaman Kursus Lagi	1000
8	Akses Halaman Diskusi	1000
9	Akses Form Reply Diskusi	1000
10	POST Form Reply Diskusi	1000
11	Akses Halaman Kursus Lagi	1000
12	Akses Halaman Partisipan Kursus	1000
13	POST Logout	1000

Tabel 3. 13000 *User Request*

Kategori ini merupakan kategori terakhir dari pengujian pada penelitian ini dimana terdapat 13 aksi yang akan dijalankan dengan masing-masing *dummy user* sebanyak 1000 *user*, sehingga total *requests* yang berjalan pada kategori ini adalah **13000 *Requests***.

Alat dan Bahan

Pada penelitian kali ini, peneliti akan menggunakan salah satu dari *IaaS Service* yang populer di dunia yaitu *Digial Ocean Droplets* sebagai server yang akan diuji pada penelitian kali ini. Peneliti memilih Digital Ocean dikarenakan biaya nya yang sangat murah dalam penyewaannya sehingga peneliti dapat dengan bebas menjalankan *stress test* tanpa memikirkan biaya. Kemudian peneliti akan menggunakan Laptop milik peneliti sendiri yang akan menjalankan Apache JMeter, serta *software* lainnya yang diperlukan untuk merancang arsitektur kali ini.

Sehingga dapat disimpulkan Alat dan Bahan yang akan dipergunakan pada penelitian kali ini adalah sebagai berikut:

1. Perangkat Keras
    a) 1 Unit Laptop:

Processor: Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz
RAM: 8GB DDR4
Storage: 500GB SSD

b) Digital Ocean Droplets:
Processor: Intel 2 Core Processor
RAM: 2GB
Storage: 60GB

2. Perangkat Lunak
   a) Visual Studio Code
   b) Chrome (Internet Browser)
   c) Docker Desktop Windows & Linux
   d) Apache JMeter

Teknik Pengumpulan Data

Adapun Metode pengumpulan data yang dipakai pada penelitian ini adalah metode observasi[13]. Dengan penjabaran sebagai berikut:

1. Adapun Sumber Data yang dipergunakan dalam penelitian ini adalah Data Primer
2. Subyek atau Obyek penelitian dari data primer pada penelitian ini adalah Projek Ruang Kuliah yang akan diuji coba dengan arsitektur yang telah di rancang.
3. Dan untuk Jenis Data yang di pergunakan dalam penelitian ini adalah Data Kontinum dalam bentuk data nominal. Dikarenakan data yang didapatkan dinyatakan dalam bentuk angka-angka dari beberapa kategori kondisi pengujian.

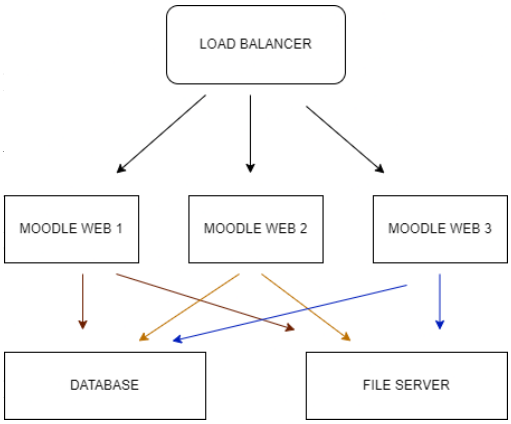
Identifikasi Masalah

Langkah pertama yang dilakukan adalah mengidentifikasi masalah untuk diteliti. Masalah yang ditemukan pada penelitian ini bersumber dari apa yang dialami penulis saat melakukan pengembangan platform Ruang Kuliah. Berdasarkan hal tersebut, disertai dengan teori yang diperoleh, penulis dapat menjabarkan masalah penelitian

Studi Literatur

Setelah mengidentifikasi masalah, dilakukan studi literatur yang bertujuan untuk mendapatkan teori-teori yang relevan untuk memperkuat argumentasi terkait permasalahan yang akan dibahas. Literatur dapat berupa buku, artikel ataupun jurnal.

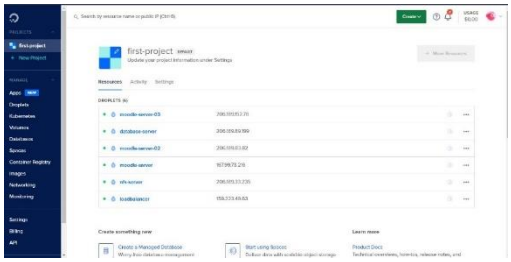
Perancangan Arsitektur



Gambar 2. Arsitektur Moodle Clustered

Berdasarkan Gambar diatas, pada penelitian ini total akan menggunakan 6 unit Droplet pada arsitektur. Dimana 3 unit Droplet sebagai *server* dari Moodle, 1 unit Droplet untuk *Database*, 1 unit Droplet untuk *File Server* dan terakhir 1 unit Droplet untuk *Load Balancer*. Dapat dilihat bahwa rancangan pada penelitian kali ini peneliti menggunakan *File Server* pada Arsitekturnya. Berdasarkan riset yang telah dilakukan secara mendalam, *File Server* disini dapat digunakan untuk berdiamnya *folder moodle_data* nantinya, sehingga *folder* ini berdiri secara terpusat dan nantinya akan di teruskan pada setiap *instans server* Moodle. Karena setiap *instans server* Moodle yang dibuat harus memiliki kondisi *moodle_data* yang sama.

Implementasi Rancangan



Gambar 3. Implementasi Rancangan Pada Digital Ocean

Setelah melakukan perancangan arsitektur maka sampailah ke tahap implementasi dari rancangan tersebut. Pada penelitian ini akan digunakan Docker sebagai teknologi yang akan di *install* kepada tiap Droplet Digital Ocean. Docker tersebut akan menyimpan konfigurasi setup untuk tiap *server*. Berikut adalah File Konfigurasi untuk masing-masing droplet *Server*.

Konfigurasi Load Balancer

```
version: ‘2’
services:
  load_balancer:
    image: nginx:alpine
    restart: unless-stopped
    ports:
      - "80:80"
    volumes:
      -
.docker/nginx/conf.d/:/etc/nginx/co
nf.d/
      -
.docker/data/nginx/:/var/log/nginx/
```

Gambar 4. Docker Compose Load Balancer

Konfigurasi yang di gunakan disini menggunakan sintaks konfigurasi Docker Compose versi ke 2. Peneliti disini menggunakan konfigurasi Docker Compose versi ke 2 dikarenakan versi 1 telah deprecated. Sintaks versi ke 2 ini di support oleh Docker Engine untuk versi 1.10.0 dan diatasnya. Pada sintaks versi ke 2 ini, setiap aplikasi atau program yang ingin dibuatkan container nya harus dibungkus didalam atribut bernama services. Pada server Load Balancer ini, terdapat satu buah service yang digunakan disini yaitu Nginx. Service Nginx disini akan di tampung kedalam suatu container dengan nama load_balancer. Lalu untuk Image yang digunakan disini adalah nginx:alpine, yang kemudian pada konfigurasinya peneliti melakukan penyesuaian kembali seperti memilih algoritma yang digunakan dan mendaftarkan server-server Moodle. Berikut adalah konfigurasinya.

```
upstream backend {
    least_conn;

    #MOODLE DROPLET PRIVATE IP
    server 10.104.0.7;
    server 10.104.0.5;
    server 10.104.0.4;
}

server {
    listen 80; #DEFAULT HTTP PORT

    location / {
       #POINTING TO BACKEND LOAD
BALANCER
       proxy_pass http://backend;

       #Adjust header for css and
js assets
       proxy_set_header Host $host;
       proxy_set_header X-
Forwarded-For $remote_addr;

    }
}
```

Gambar 5. NGINX Conf Load Balancer

Konfigurasi Database

```
version: '2'
services:
  #MySQL Service
  db:
    image: mysql:5.7
    restart: unless-stopped
    environment:
      MYSQL_DATABASE: db_moodle
      MYSQL_ROOT_PASSWORD: rootpass
    ports:
      - "3306:3306"
    volumes:
```

```
      -
.docker/data/db:/var/lib/mysql/
      -
.docker/mysql/my.cnf:/etc/mysql/my.
cnf
```

Gambar 6. Docker Compose Database Server

Untuk konfigurasi Docker Compose dari server pada database tidak terdapat perbedaan yang signifikan, menggunakan Image Official, menggunakan port default 3306 dan volumes untuk sinkronisasi data seperti server Load Balancer sebelumnya. Perbedaannya hanya pada salah satu atribut tambahan yaitu environment. Environment disini berfungsi untuk memberi tahu nama database utama dari Moodle, serta password dari yang digunakan Moodle nanti ketika mengakses database.

Konfigurasi File System

```
version: "2"
services:
  nfs:
    image: itsthenetwork/nfs-
server-alpine:12
    restart: unless-stopped
    privileged: true
    environment:
      - SHARED_DIRECTORY=/data
    volumes:
      - ./moodledata:/data
    ports:
      - 2049:2049
```

Gambar 7. Docker Compose NFS Server

Untuk konfigurasi pada server NFS juga memiliki konfigurasi Docker Compose yang tidak berbeda dari sebelumnya. Pada server ini menggunakan Image yang telah di sediakan oleh komunitas, dan menggunakan port default dari NFS. Lalu pada bagian environment kita memberi tahu NFS bahwa folder virtual didalam container yang akan digunakan sebagai tempat berdiamnya data berada pada path /data. Lalu folder virtual /data tersebut akan di sinkronisasikan dengan folder pada Host Droplet yaitu folder moodledata. Nantinya folder moodledata inilah yang akan di akses oleh ketiga server Moodle.

Konfigurasi Moodle Server

Setelah menyiapkan semua server yang dibutuhkan moodle, sampailah pada tahap menyiapkan server moodle nya. Disini moodle diarahkan pada server db, dan folder moodle data di masing-masing server moodle di arahkan ke Network File System. Berikut ini adalah konfigurasi Docker Compose nya.

```
version: '2'
services:
  moodle:
    image:
'docker.io/bitnami/moodle:3-debian-
10'
    ports:
      - 80:8080
    environment:
      - MOODLE_DATABASE_TYPE=mysqli
      -
MOODLE_DATABASE_HOST=10.104.0.3
      - MOODLE_DATABASE_USER=root
      -
MOODLE_DATABASE_NAME=db_moodle
      -
MOODLE_DATABASE_PASSWORD=rootpass
      -
MOODLE_DATABASE_PORT_NUMBER=13306
      - MOODLE_USERNAME=admin
      -
MOODLE_PASSWORD=adminprimakara
      - MOODLE_SITE_NAME=Ruang
Kuliah
      - BITNAMI_DEBUG=true
    volumes:
      -
./moodle_app:/bitnami/moodle
      -
./moodle_app_data:/bitnami/moodleda
ta
```

Gambar 8. Docker Compose Moodle Server

Kemudian jalankan perintah berikut untuk menyambungkan folder data dari moodle ke server NFS.

```
sudo apt install nfs-client -y
sudo mount -v -o vers=4,loud
10.104.0.2:/ ~/moodle-
```

```
cluster/3_moodle_server/moodle_app_
data
```

Hasil Pengujian

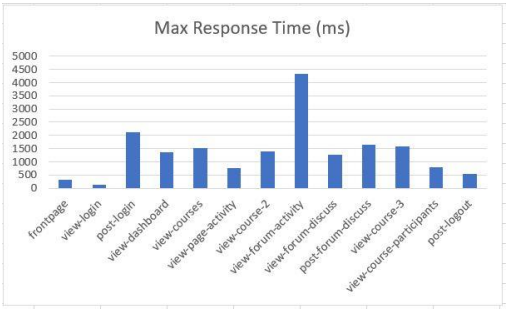
Setelah berhasil menyelesaikan konfigurasi *setup server* dari arsitektur Moodle Clustered dengan menggunakan Nginx sebagai *Load Balancer*, maka sampailah ke tahap dimana peneliti melakukan pengujian. Peneliti disini akan menguji arsitektur dengan kategori *390 requests*, *1300 requests* dan *13000 requests*. Kemudian hasil pengujian dari arsitektur ini akan di bandingkan dengan arsitektur Moodle yang hanya menggunakan 1 unit Droplet berisi *Database* Bersama dengan Moodle dalam 1 *server*. Berikut adalah penjabaran hasil pengujiannya.

Rancangan	Max Response Time (ms)		
	30 User (390 Reqs)	100 User (1300 Reqs)	1000 User (13000 Reqs)
Single Server Moodle	3191ms	21678ms	102630ms
Moodle Clustered	4329ms	5950ms	28504ms

Tabel 4. Hasil Pengujian Arsitektur

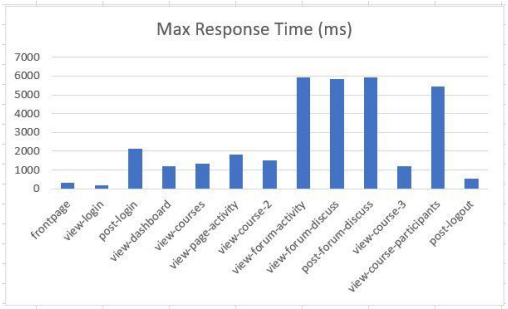
Seperti yang tertera pada Tabel, untuk kategori jumlah *user* yang sedikit yaitu sebanyak **390 *Requests***, melakukan *Load Balancing* dengan algoritma *Least Connection* ternyata tidak memberikan perbedaan kecepatan *response* yang jauh dari rancangan yang hanya menggunakan ***Single Server***. Selisih keduanya hanya **1138ms**, atau sekitar **1 detik** pada saat diuji dengan jumlah *request* yang sedikit. Akan tetapi, ketika menyentuh **1300 *Requests*** dan **13000 *Requests***, melakukan *Load Balancing* benar-benar terlihat efeknya. Ketika diuji oleh 1300 *Requests*, *Load Balancing* dapat memberikan kecepatan *Response Time* paling berat hanya menyentuh **5950ms**, dibandingkan ketika masih *Single Server* yang mencapai **21678ms**.

Lalu untuk 13000 *Requests*, *Response Time* paling berat dari kedua rancangan juga mengalami perbedaan yang sangat jauh. Namun, hasil yang diberikan oleh rancangan Moodle Clustered masih terbilang cukup tinggi yaitu sebesar **28504ms**, atau sekitar **28 detik**. Namun perlu diingat bahwa pada pengujian ini tiap aksi yang dijalankan memiliki tingkat kompleksitas yang berbeda-beda. Sehingga peneliti kemudian melakukan pengecekan mendalam dan melakukan perbandingan kecepatan response untuk tiap aksi pada setiap kategori *requests*. Berikut penjabarannya.



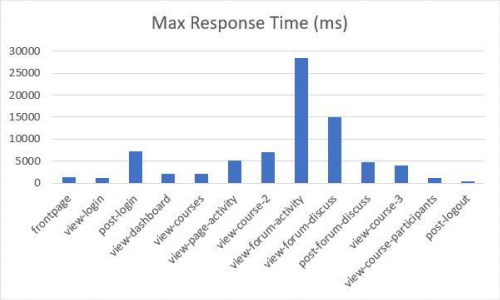
Gambar 9. Max Response Time Kategori 390 Requests

Untuk kategori pertama dengan jumlah *requests* 390, aksi terberat terjadi pada aksi ”*Akses Halaman Activity*” yang mencapai **4329ms**.



Gambar 10. Max Response Time Kategori 1300 Requests

Kemudian untuk kategori kedua dengan jumlah 1300 *Requests*, pada pengujian kali ini ternyata terdapat tiga aksi yang *response time* nya diatas **5000ms.** Yaitu “*Akses Halaman Activity*”, “*Akses Form Reply Diskusi*”, dan yang terakhir “*POST Form Reply Diskusi*”. Ketiga aksi tersebut memiliki *Response Time* yang beda tipis, dimana yang paling tinggi yaitu **5950ms** untuk aksi “*Akses Halaman Activity*”.



Gambar 11. Max Response Time Kategori 13000 Requests

Kemudian untuk kategori ketiga dengan jumlah 13000 *Requests*, pada pengujian kali ini tiap aksi memiliki *Response Time* yang berbeda-beda. Dan untuk *Response Time* paling tinggi hingga **28504ms** di capai oleh aksi "*Akses Halaman Activity*", sama seperti sebelumnya yang terjadi pada kategori pengujian kedua. Sehingga dengan hasil pengujian ini terbukti bahwa melakukan *Load Balancing* memberikan efek jika dibandingkan dengan Single Server. Namun, ketika requests diatas 10000 *requests* arsitektur kali ini masih **belum** memberikan hasil yang maksimal, terutama saat melakukan akses halaman *activity* pada Moodle.

SIMPULAN

Platform Ruang Kuliah dibangun menggunakan Moodle *LMS*. Sehingga pada penelitian ini dilakukan pengujian menggunakan *sample data* dalam bentuk Apache JMeter *Scripts* yang telah disediakan secara *official* pada pihak Moodle. Pengujian menggunakan tiga kategori jumlah *user* yang disediakan oleh pihak Moodle, dengan masing-masing melakukan eksekusi total 13 aksi. Yaitu kategori pertama dengan dengan total 390 *request*. Kategori kedua dengan total 1300 *request*. Kategori ketiga dengan total 13000 *request*. Penelitian pun dilakukan dengan mencoba melakukan perancangan Arsitektur Moodle berbentuk *Cluster* yang memiliki 3 *server* Moodle dengan satu *server Load Balancer* menggunakan algorima *Least Connection*. Berdasarkan pengujian yang telah dilakukan ternyata melakukan *Load Balancing* pada Moodle *Server* untuk studi kasus kali ini dapat memberikan efek jika dibandingkan dengan *Single Server* Moodle atau tanpa *Load Balancing*. Namun, ketika jumlah *requests* diatas 10000 requests arsitektur kali ini masih belum memberikan hasil yang maksimal.

DAFTAR PUSTAKA

[1] Direktorat Jenderal Pendidikan Tinggi Kementerian Pendidikan dan Kebudayaan. 2020. Buku Panduan Merdeka Belajar Kampus Merdeka. Kata Pengantar Hal 1-2.

[2] Ben Young. eLearning Industry. 2018. Top 3 Advantages Of Moodle. [Online]. https://elearningindustry.com/advantages-of-moodle-top-3. [Accessed: 28 December 2021]

[3] Afdhal. 2013. Studi Perbandingan Layanan Cloud Computing. Jurusan Teknik Elektro, Fakultas Teknik, Universitas Syiah Kuala

[4] Sulastri Apridayanti, Isnawaty, Rizal Adi Saputra. 2018. "Desain dan Implementasi Virtualisasi Berbasis Docker untuk Deployment Aplikasi Web". Universitas Halu Oleo.

[5] Rakhmi Khalida, Adi Muhaijrin, Siti Setiawati. 2019. "Teknis Kerja Docker Container untuk Optimalisasi Penyebaran Aplikasi". Universitas Bhayangkara Jakarta Raya.

[6] G2. [Online]. Available: https://www.g2.com/categories/infrastructure-as-a-service-iaas. [Accessed: 10 May 2021].

[7] IBM. 2007. Virtualization in Education. The Greaves Group, 4, 3-4..

[8] Russ McKendrick. 2020. Mastering Docker Fourth Edition, Chapter 1 Docker Overview.

[9] Khaja Saik. Scalability: Horizontal vs Vertical Scaling. [Online]. Available: https://www.linkedin.com/pulse/scalability-horizontal-vs-vertical-scaling-scale-outin-khaja-shaik/. [Accessed: 20 June 2021].

[10] Surya Begum. 2021. Load Balancing Algorithms in Cloud Computing Systems. LAP LAMBERT Academic Publishing.

[11] NGINX Docs. HTTP Load Balancing. [Online]. Available: https://docs.nginx.com/nginx/admin-

guide/load-balancer/http-load-balancer/. [Accessed: 28 December 2021].
[12] AWS Editorial Team. 2019. AWS Ops Automator v2 features vertical scaling. [Online]. Available: https://aws.amazon.com/blogs/architecture/aws-ops-automator-v2-features-vertical-scaling-preview/. [Accessed: 20 June 2021].
[13] Widoyoko, Eko Putro. (2014). Teknik Penyusunan Instrumen Penelitian. Yogyakarta: Pustaka Pelajar