# Implementasi Dan *Profiling* Layanan Cms Wordpress Menggunakan Fitur *Horizontal Pod Autoscaler* Pada Google Kubernetes Engine Dengan Metrik Cpu, *Pod*, Dan *Transaction*

1st Arya Bimo Bagas Penggalih
*Fakultas Rekayasa Industri*
*Universitas Telkom*
Bandung, Indonesia
aryabimo@student.telkomuniverity.ac.id

2nd Adityas Widjajarto
*Fakultas Rekayasa Industri*
*Universitas Telkom*
Bandung, Indonesia
adtwjrt@telkomuniversity.co.id

3rd Avon Budiyono
*Fakultas Rekayasa Industri*
*Universitas Telkom*
Bandung, Indonesia
avonbudi@telkomuniversity.ac.id

***Abstrak*— Aspek skalabilitas layanan pada *cloud platform* dapat dilakukan dengan *container* manajemen dengan Kubernetes Berdasarkan hal tersebut, penggunaan fitur pada Google Kubernetes Engine yaitu *horizontal pod autoscaler* dapat menjadi solusi. Metode yang digunakan pada pengujian ini menggunakan *Load Testing* untuk mengukur utilisasi CPU, dan kemampuan layanan menerima *request* dari varian jumlah *user* tersebut. Pengujian pada layanan disimulasikan menggunakan platform *cloud* yaitu Google Cloud Platform dengan *service* Google Kubernetes Engine. Pada layanan CMS yang menggunakan HPA memiliki konfigurasi CPU 70%. Rata-rata penggunaan CPU dibawah 70% masih termasuk batas aman. Dari hasil analisis penerimaan *request* tidak ada perbedaan yang signifikan pada *total request* dan average response time. Namun pada parameter persentase *failure* layanan CMS yang menggunakan HPA, terdapat perbedaan persentase *failure* yang cukup tinggi. Seperti pada varian user 200 layanan CMS Non-HPA memiliki *total request* 2161 dengan persentase *failure* 100% dan layanan CMS dengan HPA memiliki *total request* 1578 dengan persentase *failure* 14%. Dengan persentase *failure* yang lebih kecil dapat di indikasikan HPA dapat menjadi salah satu solusi meningkatkan skalabilitas.**

***Kata kunci:* Kubernetes, *Load Testing*, *Horizontal Pod Autoscaler*, skalabilitas**

## I. PENDAHULUAN

Pada era modern komputasi awan dan teknologi kontainerisasi, Kubernetes telah menjadi salah satu *platform* orkestrasi kontainer paling populer yang digunakan untuk mengelola aplikasi skala besar. Penggunaan Kubernetes memungkinkan pengembang layanan untuk dengan mudah menyebarkan dan mengelola aplikasi di lingkungan yang kompleks dan dinamis. penggunaan aplikasi berbasis web cenderung meningkat seiring banyaknya pengguna internet diseluruh dunia. Dari data statistik pengguna internet secara individual menurut international Telecomunication Union (ITU) pada paper 2019 Measuring digital development Facts and figures 2019, pada tahun tersebut rata-rata pengguna internet individualnya mencapai 53.6% diseluruh dunia.[1].
Mulai banyaknya permintaan *request* jumlah pengguna internet menjadikan penyedia layanan atau perusahaan aplikasi berbasis *web* harus meningkatkan dari segi aspek skalabilitas mereka terutama *server*, semakin meningkat jumlah permintaan *request* maka akan mengakibatkan *server down* atau *down time* karena *server overload*.

Teknologi Cloud computing dibutuhkan sebagai teknologi komputasi terdistribusi yang mampu mengabstraksikan kemampuan *hardware* atau perangkat keras dalam menjalankan sebuah proses komputasi atau sistem operasi, efisiensi *resource* penggunaan perangkat dari segi *hardware* dan proses untuk high availability, serta sistem yang lebih handal untuk menangani sistem kegagalan.[2].

Penerapan *auto scaling* dapat mengoptimalkan layanan *cloud* yang bersifat *on-demand service* dan *rapid elasticy* dengan metode pengelolaan secara otomatis yang dilakukan oleh sistem. Salah satu metode *auto scaling* yaitu dengan memprediksi penggunaan *resource* di sisi *server* sehingga sistem *cloud* yang dibangun dapat memperkirakan berapa jumlah *resource* yang dibutuhkan oleh layanan[3].

*Horizontal Pod Autoscaling* (HPA) adalah fitur penting dalam Kubernetes yang memungkinkan infrastruktur secara otomatis menyesuaikan jumlah replika *pod* berdasarkan permintaan lalu lintas dan penggunaan sumber daya. Penggunaan HPA pada Google Kuberntes Engine (GKE) memungkinkan aplikasi untuk meningkatkan jumlah replika *pod* saat beban kerja meningkat dan mengurangi jumlah replika *pod* saat beban kerja menurun, secara otomatis mengoptimalkan penggunaan sumber daya dan meningkatkan elastisitas aplikasi.[4].

## II. DASAR TEORI

### A. Central Processing Unit (CPU)

CPU (central processing unit) atau sering disebut juga dengan istilah processor, adalah bagian dari sebuah sistem komputer yang melakukan instruksi dari program komputer, dan merupakan unsur utama yang melaksanakan fungsi komputer. Processor juga seringkali disebut sebagai otak dari sebuah komputer.Utilisasi CPU adalah penggunaan sumber daya komputasi yaitu CPU. Utilisasi CPU memiliki *Key*

*Performance Indicator* sebagai standar batas utilisasi dengan jika penggunaan lebih dari 70% masuk kedalam *Warning threshold*, jika penggunaan lebih dari 90% masuk kedalam *Critical Threshold*.[5]

B. *Skalabilitas*

skalabilitas adalah kemampuan sistem untuk menyesuaikan masalah dalam ruang lingkup masalah peningkatan, seperti peningkatan jumlah volume pekerjaan yang dilakukan sistem.[6].

C. *Load Testing*

*Load testing* merupakan bentuk pengujian yang bertujuan untuk mengevaluasi kemampuan suatu sistem, aplikasi, atau infrastruktur dalam menangani beban kerja yang tinggi atau volume pengguna yang besar. Tujuan utama dari *Load testing* adalah untuk menemukan batas kinerja sistem, mengidentifikasi *bottleneck*, dan memastikan bahwa aplikasi berjalan dengan lancar di bawah tekanan maksimal.

D. Google Cloud Platform

Merupakan produk yang dikembangkan oleh Google yang dimana berfokus pada layanan cloud computing untuk membuat, menguji, dan men-deploy dengan menggunakan infrastruktur dan fitur Google yang dapat diskalakan.

*E. Virtual Machine*

*Virtual machine* (VM) adalah suatu teknologi yang memungkinkan replikasi mesin fisik dalam bentuk mesin virtual yang dapat menjalankan sistem operasi dan aplikasi secara terisolasi.

F. Locust

Locust adalah sebuah perangkat lunak (*software*) open-source yang digunakan untuk melakukan uji beban (*Load testing*) terhadap sistem atau aplikasi. Dalam konteks tinjauan pustaka, Locust dapat digunakan untuk menguji performa dan keandalan sebuah aplikasi atau sistem yang terkait dengan penelitian yang sedang dilakukan.

G. Kubernetes

Kubernetes adalah platform *open source* untuk mengelola kumpulan kontainer dalam suatu *cluster server*. Platform ini pertama kali dikembangkan oleh Google dan kini dikelola oleh Cloud Native Computing Foundation (CNCF) sebagai platform manajemen kontainer yang cukup populer. Kontainer sendiri adalah *environment* dengan sumber daya, CPU, dan sistem file untuk satu aplikasi. Jadi, aplikasi tersebut akan memiliki sumber daya sendiri. Keuntungannya, aplikasi jadi tidak mudah mengalami *downtime*

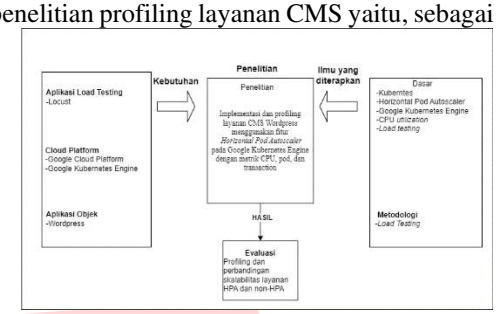
H. Horizontal Pod Autoscaler

Horizontal Pod Autoscaling (HPA) adalah mekanisme otomatis dalam Kubernetes yang digunakan untuk menyesuaikan jumlah replika *pod* secara *horizontal* berdasarkan beban kerja atau tingkat penggunaan sumber daya. HPA dapat digunakan untuk mengoptimalkan penggunaan sumber daya.

## III. METODELOGI PENELITIAN

A. Model Konseptual Penelitian

Model konseptual ini bertujuan untuk memudahkan dalam melakukan identifikasi permasalahan yang ditemukan pada penelitian profiling layanan CMS yaitu, sebagai berikut:

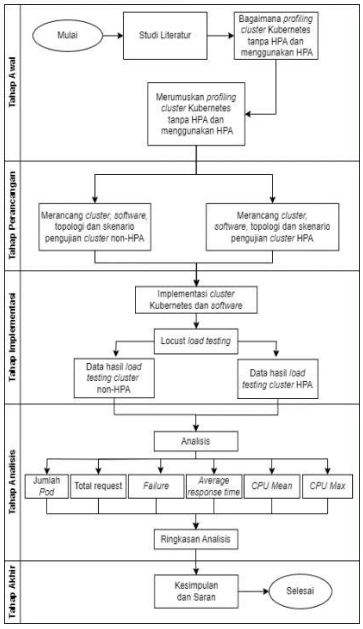



GAMBAR 1
Model Konseptual Penelitian

Dapat diketahui pada Gambar 1 bahwa terdapat tiga ruang lingkup yaitu kebutuhan, penelitian dan dasar ilmu yang di terapkan. Pada aspek Kebutuhan terdiri dari 3 bagian diantaranya aplikasi *load testing* yaitu *platform* pengujian yaitu Google Kubernetes Engine dan aplikasi objek yaitu layanan CMS. Ilmu yang diterapkan terdiri dari dasar teori dan metodologi. Dasar Teori yang digunakan diantaranya Kubernetes, *Horizontal Pod Autoscaler,* CPU *Utilization* dan *Load Testing*.

B. Sistematika Penelitian

Dalam melakukan penelitian ilmiah, sistematika penelitian diperlukan agar penelitian yang dilaksanakan berjalan terstruktur dan sistematis. Penelitian diawali dengan tahap identifikasi masalah, analisis, desain, eksperimen, dan terakhir evaluasi dan kesimpulan




GAMBAR 2
Sistematika Pen*yelesaian Masalah*

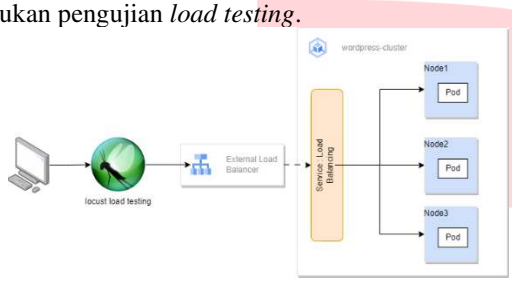
## IV. HASIL DAN PEMBAHASAN

### A. Rancangan Sistem

Dalam Melakukan eksperimen load testing pada CMS dibutuhkannya penracangan sistem untuk melakukan pengujian dengan menggunakan beberapa instrument eksperimen. Instrumen yang digunakan berguna untuk mendukung jalannya eksperimen. Maka dari itu, dilakukan identifikasi instrument eksperimen yang terdiri dari intriment fisik dan program. Berikut adalah instrument eksperimen yang digunakan.

1. Topologi Jaringan

Topologi jaringan digunakan untuk menggambarkan kegiatan yang akan dilakukan dalam pengujian ini, sebagai langkah untuk mendapatkan hasil percobaan dalam melakukan pengujian *load testing*.




GAMBAR 3
Topologi Jaringan Pengujian

2. Daftar IP Address

IP *Address* yang digunakan pada penelitian ini, dilampirkan pada Tabel 1 Daftar IP address:

*TABEL 1*
*Daftar IP Address*

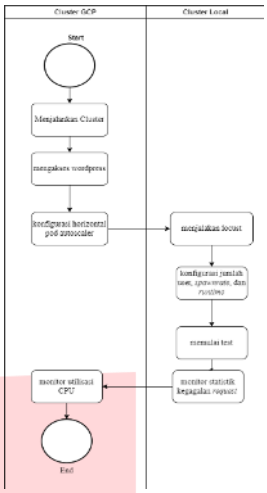
| Host | IP Address |
|---|---|
| Laptop | 192.168.100.48 |
| Wordpress | 34.110.182.68 |
| *Pod* | 10.116.9.62/* |
| Locust | 0.0.0.0:8089 |

### B. Skenario Pengujian

Dalam skenario pengujian ini, locust yang di *install* pada cluster *local* melakukan *load test* dengan varian simultan *user* yaitu 20, 40, 60, 80, 100, 120, 140, 180, dan 200 dengan waktu uji coba 1 menit untuk setiap varian simultan *user* dengan *task* melakukan *request* "/" dan Post_id kepada aplikasi objek yang di *deploy* pada aplikasi Web CMS yang di deploy platform Kubernetes di Google Cloud Platform.

1. Skenario 1: pengujian layanan CMS non-HPA

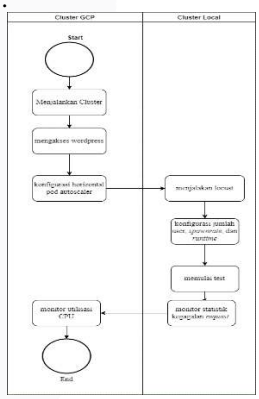
Dalam skenario pengujian ini, locust yang di install pada *cluster local* melakukan *load test* dengan varian simultan user yaitu 20, 40, 60, 80, 100, 120, 140, 180, dan 200 dengan waktu uji coba 1 menit untuk setiap varian simultan *user* kepada aplikasi objek yang di *deploy* pada platform Kubernetes di Google Cloud Platform tanpa menggunakan fitur *Horinzontal autoscaler*




GAMBAR *4*
Pengujian Load Test *non-HPA*

2. Skenario 2: Pengujian layanan CMS HPA

Dalam scenario pengujian ini locust digunakan untuk mengukur kemampuan layanan CMS yang sudah menggunakan HorizontalPodAutoscaler dengan konfigurasi utilisasi CPU 70% untuk menambahkan replika minimal 3 pod dan maksimal 30 pod dengan varian simultan user yang digunakan untuk pengujian ini yaitu 20, 40, 60, 80, 100, 120, 160, 180, dan 200 dengan waktu uji coba 1 menit untuk setiap varian simultan user.




GAMBAR 5
Pengujian *Load test* HPA

### C. Pengujian

Pada tahapan ini akan dilakukan 2 pengujian dimulai dengan pengujian load testing pada layanan CMS non-HPA lalu dilanjutkan dengan pengujian load testing pada layanan CMS HPA

1. Pengujian Layanan CMS non-HPA

Dalam spengujian ini, menggunakan Locust unutk melakukan *load test* dengan varian simultan user yaitu 20, 40, 60, 80, 100, 120, 140, 180, dan 200 dengan waktu uji coba 1 menit untuk setiap varian simultan *user* kepada aplikasi objek yang di *deploy* pada platform Kubernetes di Google Cloud Platform tanpa menggunakan fitur *Horinzontal autoscaler*.

a. Setelah aplikasi objek berhasil di deploy kedalam Google Kubernetes Engine maka perlu mengakses ke external ip aplikasi objek untuk memastikan bahwa aplikasi objek sudah berhasil di deploy.

b. Menjalankan locust yang sudah terisntall di dalam *cluster* local dan membuka locust interface dibrower.
c. Setelah mengakses locust interface, melakukan konfigurasi dengan memasukan jumlah user, spawnrate, runtime,dan external ip aplikasi objek.
d. Setelah melakukan konfigurasi dapat langsung menjalankan *Load testing* kepada aplikasi objek
e. Setelah *Load testing* sudah berjalan lakukan monitoring statistic kegagalan *request* yang dapat di monitor dengan locust interface yang sudah di jalankan di *cluster* local.
f. Setelah loadtesting selesai maka dapat dilanjutkan memonitor CPU yang dapat di monitor pada fitur google cloud monitoring.

2. Pengujian Layanan CMS HPA

Dalam pengujian ini, menggunakan Locust unutk melakukan *load test* dengan varian simultan user yaitu 20, 40, 60, 80, 100, 120, 140, 180, dan 200 dengan waktu uji coba 1 menit untuk setiap varian simultan *user* kepada aplikasi objek yang di *deploy* pada platform Kubernetes di Google Cloud Platform yang menggunakan fitur *Horinzontal autoscaler*.

a. Menjalanlan *cluster* pada Google Kubernetes engine yang berfungsi sebagai tempat aplikasi objek yaitu wordpress di *deploy*.
b. Setelah aplikasi objek berhasil di *deploy* kedalam Google Kubernetes Engine maka perlu mengakses ke external ip aplikasi objek untuk memastikan bahwa aplikasi objek sudah berhasil di deploy.
c. Melakukan configurasi *Horizontal Pod Autoscaler* pada aplikasi objek yaitu Wordpress yang di *deploy* di cluster Google Cloud Platform dengan konfigurasi jumlah minimal pod 3 dan maksimal pod 30.
d. Menjalankan locust yang sudah terisntall di dalam cluster local dan membuka Locust *interface* dibrowser.
e. Setelah mengakses Locust *interface*, melakukan konfigurasi dengan memasukan jumlah *user*, *spawnrate*, *runtime*,dan *external ip* aplikasi objek.
f. Setelah melakukan konfigurasi dapat langsung menjalankan *load testing* kepada aplikasi objek
g. Setelah *load testing* sudah berjalan lakukan *monitoring statistic* kegagalan *request* yang dapat di monitor dengan Locust *interface* yang sudah di jalankan di *cluster local*.
h. Setelah loadtesting selesai maka dapat dilanjutkan memonitor CPU yang dapat di monitor pada fitur google cloud monitoring.

D. Hasil Pengujian

Berikut adalah hasil pengujian *load testing* menggunakan Locust pada layanan CMS yang tidak menggunakan *Horizontal Pod Autoscaling* dan layanan CMS yang menggunakan *Horizontal Pod Autoscaling* Adapun hasil pengujian berupa parameter aspek skalabilitas seperti CPU *mean*, CPU *max*, *average response time*, *total request*, dan *failure*.

1. Hasil Pengujian Layanan CMS non-HPA

Berikut adalah hasil pengujian *load testing* menggunakan Locust pada layanan CMS yang tidak menggunakan *Horizontal Pod Autoscaling* Adapun hasil pengujian berupa parameter aspek skalabilitas seperti CPU *mean*, CPU *max*, *average response time*, *total request*, dan *failure*.

a. CPU mean

TABEL 2
Hasil Pengujian CPU *mean*

| user | Non HPA CPU Mean value (%) |
|---|---|
| 20 | 78,23 |
| 40 | 206,30 |
| 60 | 434,85 |
| 80 | 295,90 |
| 100 | 143,20 |
| 120 | 238,79 |
| 140 | 551,59 |
| 160 | 874,40 |
| 180 | 755 |
| 200 | 519,26 |

b. CPU max

TABEL 3
hasil Pengujian CPU Max

| user | Non HPA CPU Max value (%) |
|---|---|
| 20 | 282,09 |
| 40 | 706,86 |
| 60 | 690,58 |
| 80 | 531,83 |
| 100 | 381,92 |
| 120 | 498,78 |
| 140 | 1211,08 |
| 160 | 967,24 |
| 180 | 1091,74 |
| 200 | 1026,91 |

C. Average responsetime

TABEL 4
Hasil Pengujian *Average Response Time*

| user | Non HPA average response time(s) |
|---|---|
| 20 | 0,9 |
| 40 | 1,6 |
| 60 | 1,9 |
| 80 | 4,7 |
| 100 | 5,8 |
| 120 | 7,8 |
| 140 | 5,1 |
| 160 | 5,7 |
| 180 | 4,1 |
| 200 | 0,2 |

d. Total request

TABEL 5
Hasil Pengujian *Total Request*

| user | Non HPA *total request* |
|---|---|
| 20 | 418 |
| 40 | 790 |
| 60 | 930 |
| 80 | 703 |
| 100 | 759 |
| 120 | 708 |
| 140 | 1145 |
| 160 | 1142 |
| 180 | 1627 |
| 200 | 2161 |

e. failure

TABEL 6
Hasil Pengujian *Failure*

| user | *Failure* (%) |
|---|---|
| 20 | 0 |
| 40 | 0 |
| 60 | 0 |
| 80 | 6 |
| 100 | 7 |
| 120 | 8 |
| 140 | 22 |
| 160 | 32 |
| 180 | 56 |
| 200 | 100 |

2. Hasil Pengujian Layanan CMS HPA

Berikut adalah hasil pengujian *load testing* menggunakan Locust pada layanan CMS yang menggunakan *Horizontal Pod Autoscaling* Adapun hasil pengujian berupa parameter aspek skalabilitas seperti CPU *mean*, CPU *max*, *average response time*, *total request*, dan *failure*.

a. CPU mean

TABEL 7
Hasil Pengujian CPU *mean*

| user | HPA CPU mean |
|---|---|
| 20 | 50,24% |
| 40 | 53,32% |
| 60 | 62,25% |
| 80 | 64,96% |
| 100 | 69,92% |
| 120 | 42,15% |
| 140 | 58,85% |
| 160 | 64,71% |
| 180 | 62,24% |
| 200 | 70,25% |

b. CPU *max*

TABEL 8
Hasil Pengujian CPU *max*

| User | CPU Max |
|---|---|
| 20 | 147,13% |
| 40 | 147,13% |
| 60 | 119,02% |
| 80 | 92,98% |
| 100 | 175,21% |
| 120 | 83,44% |
| 140 | 98,08% |
| 160 | 162,80% |
| 180 | 91,72% |
| 200 | 122,89% |

c. *Average Response time*

TABEL 9
Hasil Pengujian *Average Response Time*

| user | HPA average response time |
|---|---|
| 20 | 0,7 |
| 40 | 0,8 |
| 60 | 0,8 |
| 80 | 1,7 |
| 100 | 2,4 |
| 120 | 2,9 |
| 140 | 3,3 |
| 160 | 4,6 |
| 180 | 4,8 |
| 200 | 5,2 |

d. *Total Request*

TABEL 10
Hasil Pengujian *Total Request*

| user | HPA *total request* |
|---|---|
| 20 | 431 |
| 40 | 616 |
| 60 | 867 |
| 80 | 1296 |
| 100 | 1333 |
| 120 | 1422 |
| 140 | 1584 |
| 160 | 1258 |
| 180 | 1453 |
| 200 | 1578 |

e. *Failure*

TABEL 11
Hasil Pengujian *Failure*

| user | HPA *failure* |
|---|---|
| 20 | 0% |
| 40 | 0% |
| 60 | 0% |
| 80 | 0% |
| 100 | 0% |
| 120 | 0% |
| 140 | 0% |
| 160 | 0% |
| 180 | 6% |
| 200 | 14% |

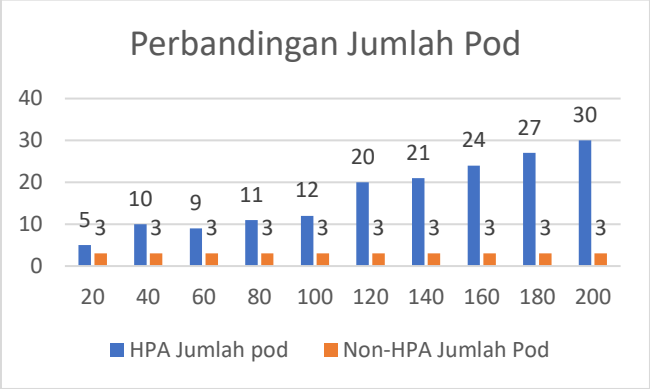
## V. ANALISIS

Selanjutnya melakukan analisis terhadap data dari layanan CMS yang menggunakan HPA dan tanpa HPA. Pada tahap ini dilakukan identifikasi untuk mengambil informasi dari data hasil pengujian yang diperoleh. Tujuan dari analisis adalah untuk mendapatkan profil atau karakter dari fungsi HPA pada cluster Kubernetes di Google Kubernetes Engine.

A. Analisis Jumlah Pod

Berikut adalah hasil pengujian *Load testing* dengan varian jumlah simultan user 20 hingga 200 jumlah replika pod pada layanan CMS Non-HPA

1. Perbandingan Jumlah Pod




GAMBAR 4
Perbandingan jumlah *pod*

Analisa perbandingan ini menunjukkan:

a. penggunaan *pod* pada non-HPA cenderung tetap
b. penggunaan *pod* pada HPA cenderung berubah

dari penelusuran data jumlah pod pada layanan diperoleh jumlah *pod* adalah:

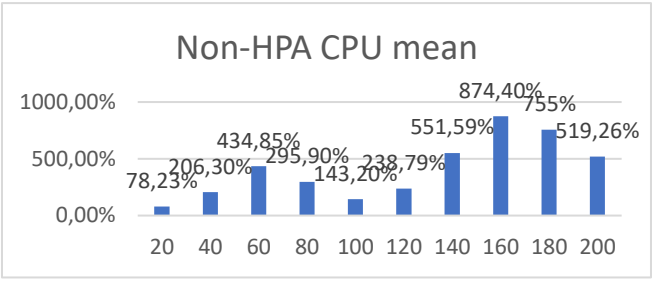
jumlah *pod* pada non-HPA sebanyak 3 *pod*.

jumlah *pod* pada HPA dapat berubah dari minimal 3 *pod* hingga maksimal 30 *pod*

B. Analsis Penggunaan CPU

Berikut ini adalah analisis CPU mean pada pengujian layanan non-HPA dan layanan dengan HPA.

1.Penggunaan CPU *mean Value* Pada Layanan CMS non-HPA




GAMBAR 5
CPU mean value non-HPA.

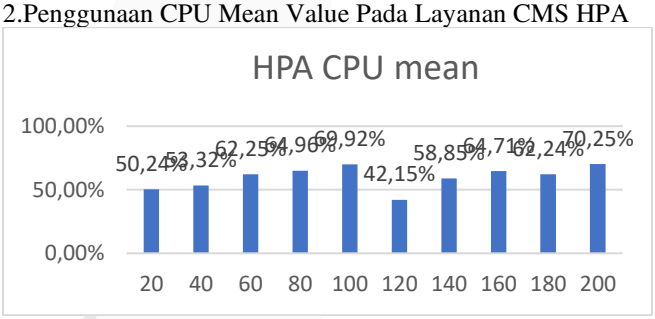
Dari analisa Penggunaan CPU *Mean* Value ini dapat diperoleh persentease penggunaan CPU *Mean* Value sebagai berikut:

a. Penggunaan CPU rata-rata tertinggi didapat pada varian jumlah user 160 dengan rata-rata penggunaan CPU 874,40%
b. Penggunaan CPU rata-rata terendah didapat pada varian jumlah user 20 dengan penggunaan CPU rata-rata 78,23%

dari penelusuran data penggunaan CPU *Mean* value Menunjukan:

a. Penggunaan CPU *Mean* Value cenderung tidak stabil
b. Penggunaan CPU *Mean* Value dapat melebihi 100%

2.Penggunaan CPU Mean Value Pada Layanan CMS HPA




GAMBAR 6
CPU *mean Value* HPA

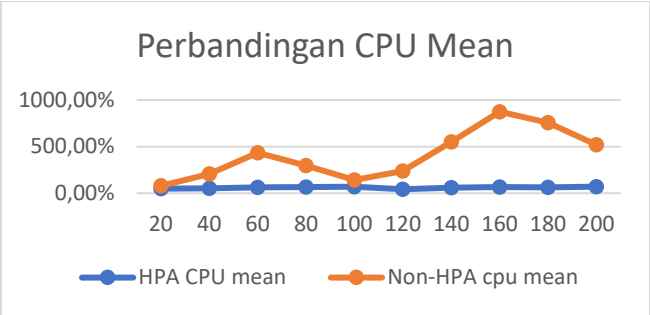
Dari analisis penggunaan CPU *Mean* Value pada Layanan CMS Dengan HPA dapat diperoleh data penggunaan CPU *mean Value* sebagai berikut:

a. Rata-rata CPU *Mean Value* tertinggi ada pada jumlah varian simultan user 200 dengan persentase CPU *Mean* Value 70,25%
b. Rata-rata CPU *Mean Value* terendah ada pada jumlah varian simultan user 120 dengan persentase CPU *Mean* Value 42,15%

Dari penelusuran data penggunaan CPU *Mean value* menunjukan:

a. Rata-rata penggunaan CPU pada layanan CMS yang menggunakan Horizontal Autoscaling dibawah 70%
b. rata-rata penggunaan CPU pada setiap varian simultan user memiliki kencenderungan naik

3. Perbandingan CPU *mean Value* Pada Layanan CMS



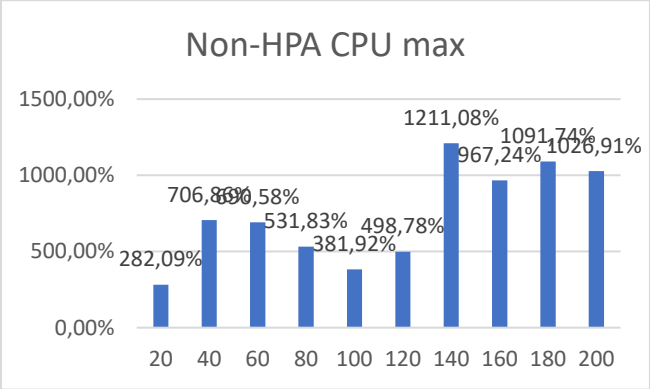

GAMBAR 7
Perbandingan CPU *mean value.*

Berikut adalah hasil analisis perbandingan dari data CPU *Mean* Value:

a. CPU *Mean* value pada layanan CMS non-HPA cenderung lebih besar
b. CPU *Mean* value pada layanan CMS HPA cenderung lebih kecil
c. CPU *Mean* Value pada layanan CMS Non-HPA cenderung tidak stabil
d. CPU *Mean* Value pada layanan CMS HPA Cenderung stabil
e. Rata-rata CPU *Mean* Value pada layanan CMS non-HPA lebih dari 100%
f. Rata-rata CPU *Mean* Value pada layanan CMS HPA dibawah 70%

.

4. Penggunaan CPU Max Pada Layanan CMS Non-HPA




Gambar 8 Hasil Pengujian Penggunaan CPU *Max* CMS Non-HPA

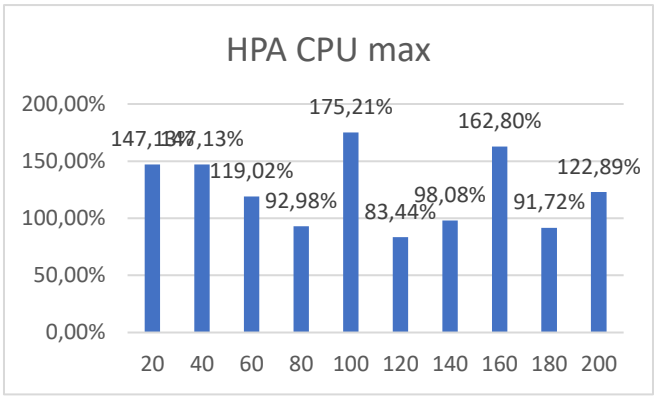
Dari analisis penggunaan CPU *Max Value* pada Layanan CMS Dengan HPA dapat diperoleh data penggunaan CPU *max Value* sebagai berikut:

a. Penggunaan CPU *Max* terkecil ada pada varian jumlah user 20 dengan persentase CPU *Max* 282,09%,
b. penggunaan CPU *Max* terbesar ada pada varian jumlah user 140 dengan penggunaan CPU *Max* 1211,08%.

Dari penelusuran data penggunaan CPU *Max* Value menunjukan:

c. CPU *Max* Value dapat lebih dari 100%
d. CPU *Max* Value memiliki kencenderungan tidak stabil

5. Penggunaan CPU *Max* Pada Layanan CMS Dengan HPA




GAMBAR 9
Hasil Pengujian Penggunaan CPU *Max* CMS HPA

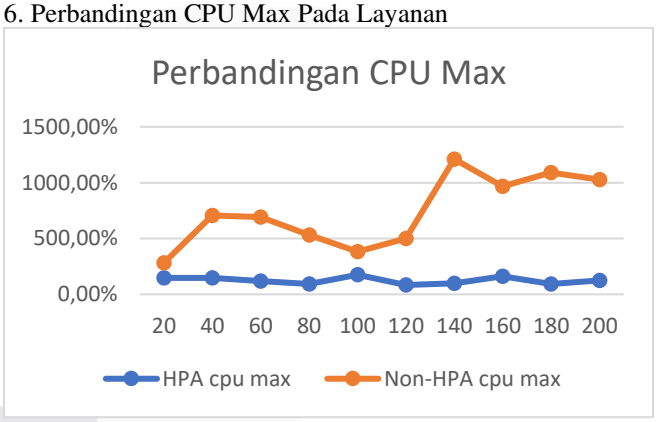
Dari analisis penggunaan CPU *max Value* pada Layanan CMS Dengan HPA dapat diperoleh data penggunaan CPU *max Value* sebagai berikut:

a. Penggunaan CPU *Max* terbesar ada pada varian jumlah user 100 dengan penggunaan CPU *Max* 175,21%.
b. Penggunaan CPU *Max* terkecil ada pada varian jumlah user 80 dengan persentase CPU *Max* 92,98%.

Dari penelusuran data penggunaan CPU *Max* Value menunjukan:

c. CPU *Max* Value dapat lebih dari 100%
d. CPU *Max* Value memiliki kencenderungan tidak stabil

6. Perbandingan CPU Max Pada Layanan




Gambar 10 Perbandingan Penggunaan CPU *Max* Pada Layanan CMS

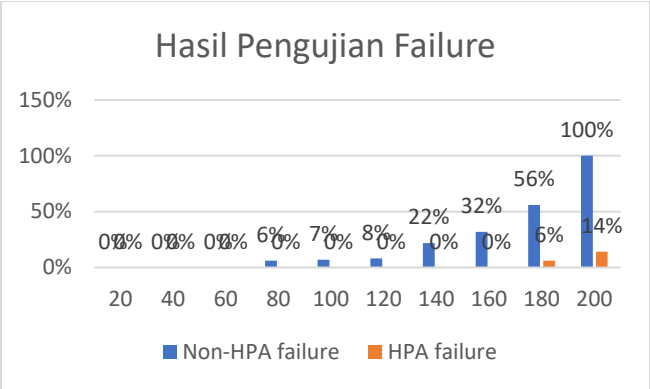
Dari hasil pengujian diperoleh ada analisis sebagai berikut:

- bahwa pada saat varian user 20 pengujian *Load testing*, penggunaan CPU *Max* pada layanan CMS yang tidak menggunakan HPA sebesar 282,09% dan layanan CMS yang menggunakan HPA sebesar 147,13%.
- Seiring berjalanya pengujian *Load testing*, penggunaan CPU *Max* pada layanan CMS yang tidak menggunakaan HPA selalu lebih besar dibanding layanan CMS yang menggunakan HPA.

C. Analisis Penerimaan Request

Pada skenario pengujian *Load testing* yang sudah dilakukan maka didapatkan data hasil dari pengujian *Load testing*. Dalam analisis. Berikut adalah hasil pengujian *Load testing* dari kedua layanan CMS dengan mengamati Penggunaan persentase *failure* pada kedua Layanan CMS:

1. Perbandingan Failure Pada Kedua Layanan CMS



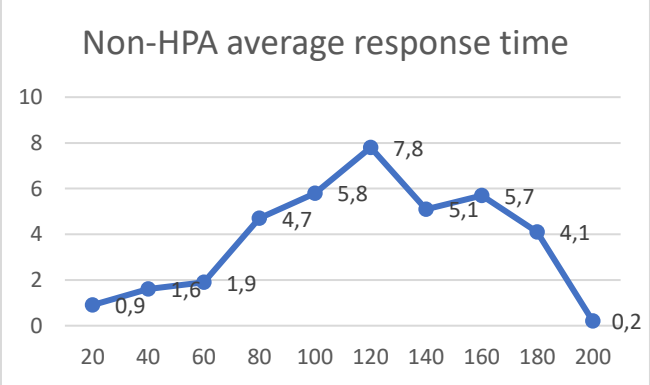

GAMBAR 11
Perbandingan *Failure* Pada Layanan CMS

Dari hasil perbandingan persentase *failure* menunjukan bahwa:

a. Layanan CMS yang menggunakan HPA memiliki persentase *failure* 0% dimulai dari varian jumlah simultan user 20 sampai 160.
b. Layanan CMS yang tidak menggunakan HPA sudah terdapat *failure* di varian jumlah simultan user 80 dengan persentase *failure* 6%.
c. Rata-rata *failure* layanan CMS non-HPA selalu lebih tinggi di bandingkan layanan CMS HPA.

dikarenakan fungsi *Horizontal Pod Autoscaling* dapat melakukan penambahan replika *pod* sesuai dengan beban yang diterima hingga jumlah maksimal konfigurasi *pod*.

2.Average Response Time Pada CMS non-HPA




GAMBAR 12
Hasil Pengujian Average Response Time CMS Non-HPA

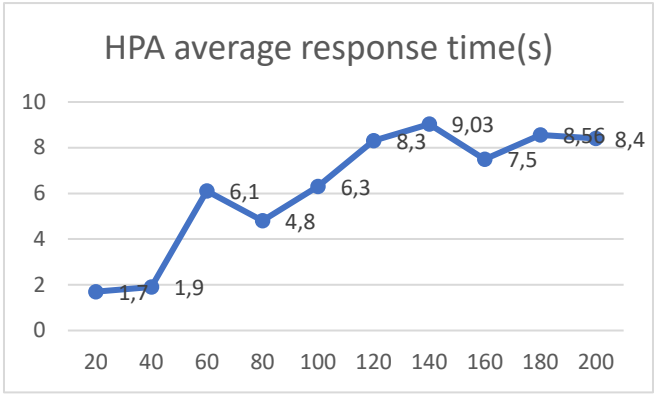
Dari hasil analisis data average response time pada layanan CMS non-HPA menunjukkan:

a. Average response time tercepat ada pada varian jumlah simultan user 200
b. Average response time terlambat ada pada varian jumlah simultan user 120

dari penelusuran data average response time pada layanan diperoleh kecenderungan sebagai berikut:

a. Pada user 20 sampai 100 average response time memiliki kecenderungan naik
b. Pada user 120 sampai 200 average response time memiliki kecenderungan turun

3. Average Response Time Pada CMS Dengan HPA

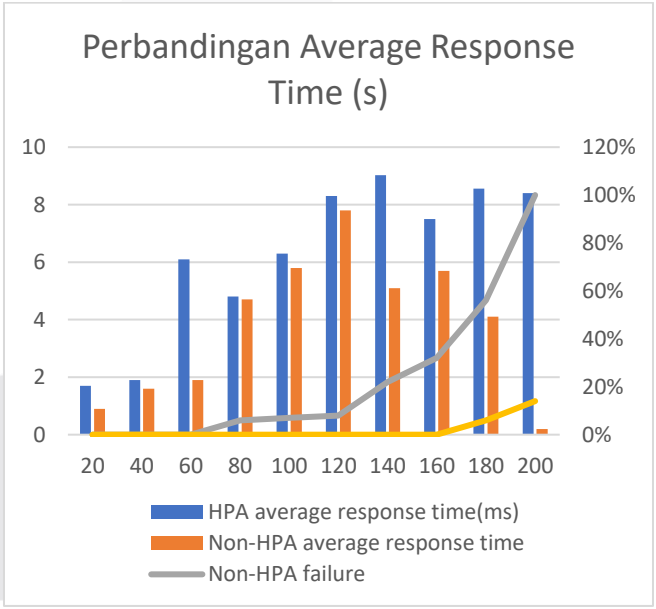



GAMBAR 13
Hasil Pengujian *Average Response Time* CMS HPA

Dari hasil analisis data *average response time* pada layanan CMS non-HPA menunjukkan:

- *Average response time* terendah ada pada user 20 dengan *average response* time 1,7 detik .
- *Average response time* tertinggi ada pada user 140 dengan average response time 9,03 detik.
- *Average response time* pada layanan CMS yang menggunakan *Horizontal Autoscaling* memiliki kecenderungan naik.

4. Perbandingan *Average Response Time* pada kedua layanan CMS




GAMBAR 14
Hasil Pengujian Average Respond time (s)

Dari hasil pengujian menunjukan data sebagai berikut:

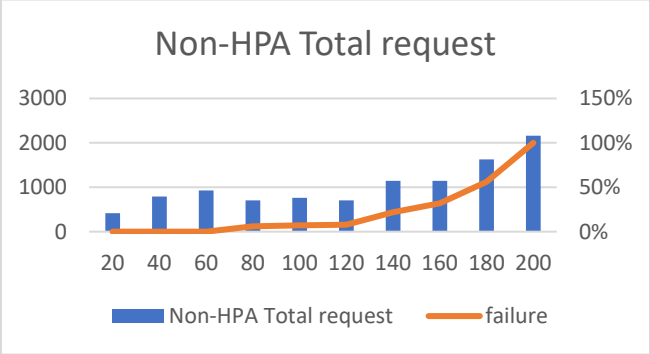
a. *Average response time* tercepat pada layanan CMS non-HPA ada pada variant user 200 dengan *average response time* 0,2 detik
b. *Average response time* terkecil pada layanan CMS HPA ada pada varian user 20 dengan *average response time* 0.9 detik
c. *Average response time* terlambat pada layanan CMS non-HPA ada pada varian user 140 dengan *average response time* 9,03 detik

d. *Average response time* terlambat pada layanan CMS HPA ada pada varian user 120 dengan *average response time* 7,8 detik

dari penelusuran data *average response time* dan *failure* pada layanan CMS diperoleh analisis perbandingan sebagai berikut:

a. *Average response time* pada layanan CMS non-HPA cenderung sedikit lebih cepat akan tetapi memiliki persentase *failure* yang lebih tinggi.
b. *Average response time* pada layanan CMS HPA cenderung sedikit lebih lambat akan tetapi memiliki persentase *failure* lebih rendah.

5. Total Request dan Failure Layanan CMS non-HPA




GAMBAR 15
Hasil Pengujian *Total request* Dan *Failure* CMS Non-HPA

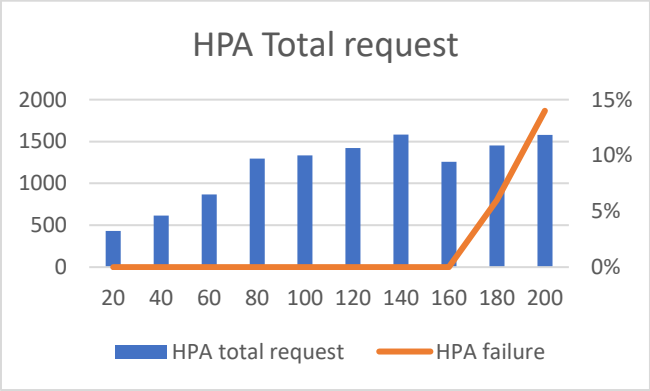
Dari hasil pengujian menunjukan data sebagai berikut:

a. *Total request* terendah dengan persentase *failure* terendah ada pada varian user 20 dengan *total request* 418 dan persentase *failure* 0%.
b. *Total request* tertinggi dengan persentase *failure* tertinggi ada pada varian user 200 dengan *total request* 2161dengan persentase *failure* 100%.

dari penelusuran data *total request* dan *failure* pada layanan CMS diperoleh analisis sebagai berikut:

a. Rata-rata *Total request* mengalami kenaikan beriringan dengan kenaikan varian jumlah simultan user.
b. *Failure* cederung naik beriringan dengan kenaikan jumlah user dan *total request.*

6. *Total Reques*t Dan *Failure* Layanan CMS Dengan HPA




GAMBAR 16
Hasil Pengujian HPA *Total request*

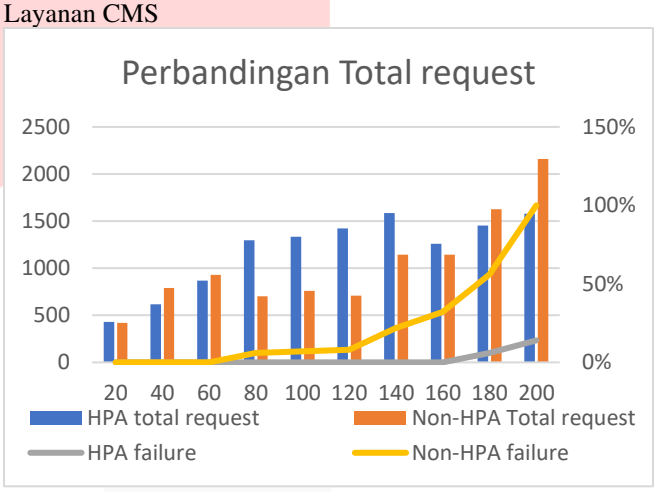
Dari hasil pengujian menunjukan data sebagai berikut:

- *Total request* terendah dengan persentase *failure* terendah ada pada varian user 20 dengan *total request* 431 dan persentase *failure* 0%.
- *Total request* tertinggi dengan persentase ada pada varian user 140 dengan *total request* 1584 dengan persentase *failure* 0%.
- *Total request* tertinggi dengan persentase *failure* tertinggi ada pada varian user 200 dengan *total request* 1578 dengan persentase *failure* 14%.

dari penelusuran data *total request* dan *failure* pada layanan CMS diperoleh analisis sebagai berikut:

c. Rata-rata *Total request* mengalami kenaikan beriringan dengan kenaikan varian jumlah simultan user.
d. *Failure* cederung naik beriringan dengan kenaikan jumlah user dan *total request*.

7. Perbandingan Total Request Dan Failure Pada Kedua Layanan CMS




GAMBAR 17
Hasil Perbandingan *Total request*

Dari hasil pengujian *Load testing* di dapatkan data bahwa pada saat pengujian *Load testing* selesai dilakukan diperoleh perbandingan sebagai berikut:

- Rata-rata *total request* layanan CMS HPA lebih banyak di banding layanan CMS non-HPA
- Persentase *failure* pada layanan CMS non-HPA lebih besar di banding layanan CMS HPA
- Pada user 180 dan 200 layanan CMS non-HPA memiliki jumlah *total request* lebih banyak dan persentase *failure* lebih tinggi.

Dengan data perbandinga hasil pengujian Load testing tersebut dapat hal ini dapat diperkirakan bahwa penggunaan HPA dapat menampung beban jumlah user request lebih banyak dikarenakan fitur HPA yang dapat menambahkan replika pod sesuai dengan beban yang di terima.maka dari itu dapat di perkirakan semakin banyak jumlah pod pada layanan CMS, maka layanan CMS dapat menerima lebih Request.

## V. KESIMPULAN

### A. Kesimpulan

Penelitian ini menghasilkan kesimpulan sebagai berikut:

1. Penerapan Horizontal Pod Autoscaling pada cluster Kubernetes dengan konfigurasi 70% dapat meningkatkan skalabilitas layanan pada cluster. Hal ini dapat dilihat dari penggunaan CPU yang stabil di

bawah 70%, serta penurunan persentase failure secara signifikan pada layanan HPA dengan persentase failure 14% dibandingkan dengan layanan non-HPA dengan persentase failure 100% pada varian jumlah user 200

2. Menguji aspek skalabilitas cluster Kubernetes dengan metode Load testing menggunakan Locust. Load testing dilakukan dengan variasi jumlah user dimulai dari 20, 40, 60, 80, 100, 120, 140, 160, 180, dan 200 user.
3. Aspek skalabilitas pada layanan yang menggunakan horizontal pod autoscaling lebih baik dibandingkan dengan layanan yang tidak menggunakan horizontal pod autoscaling. Hal ini dapat dilihat dari penggunaan rata-rata CPU yang lebih stabil di bawah 70% pada layanan yang menggunakan horizontal pod autoscaling, serta persentase failure yang lebih sedikit pada layanan CMS yang menggunakan Horizontal Pod Autoscaling dengan persentase failure tertinggi 14% pada varian jumlah user 200, dibandingkan dengan layanan CMS yang tidak menggunakan Horizontal Pod Autoscaling dengan persentase failure tertinggi 100% pada varian jumlah user 200.

B. Saran

Berdasarkan hasil analisis dan pengujian yang telah dilakukan, berikut berupa saran yang dapat disampaikan:

1. Dalam pengujian dapat menggunakan fitur *autoscaling* yang berbeda seperti *vertical pod autoscaling*
2. Dalam pengujian disarankan untuk membuat sistem dengan spesifikasi lebih tinggi dan aplikasi *Load testing* yang berbeda agar mendapatkan perbandingan yang lebih banyak sehingga informasi yang didapat lebih banyak.
3. Pada pengujian dilakukan hanya menggunakan parameter CPU, *total request*, *average response time* dan *failure*. Untuk selanjutnya dapat dilakukan pengujian menggunakan parameter uji yang lain agar dapat mendapatkan informasi dari parameter yang lainya.

.

REFERENSI


[1] ITU/UN tech agency. (2023). Measuring digital development: Facts & figures 2019. *ITU Hub*. https://www.itu.int/hub/2020/05/measuring-digital-development-facts-figures-2019/

[2] Kusuma, T. P., Munadi, R., & Sanjoyo, D. D. (2017). Implementasi dan analisis computer clustering system dengan menggunakan virtualisasi Docker. *eProceedings of Engineering*, *4*

[3] Singh, H. (2022, January 6). Understanding how Autoscaling works in DevOps - debut Infotech - medium. *Medium*. https://medium.com/debutinfotech/understanding-how-autoscaling-works-in-devops-7bb04503eff6

[4] *Horizontal pod autoscaling*. (2023, July 25). Kubernetes. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[5] Cockcroft, A. (2001). *Capacity planning for internet services: Quick Planning Techniques for High Growth Rates*.

[6] Falatah, M. M., & Batarfi, O. (2014). Cloud scalability Considerations. *International Journal of Computer Science & Engineering Survey*, *5*(4), 37–47. https://doi.org/10.5121/ijcses.2014.5403