Jurnal Teknologi Rekayasa Informasi dan Komputer Vol.7 No.1 September 2023 P-ISSN : **2581-2882**
e-ISSN : 2797-1724

Implementasi Autoscaling Untuk Meningkatkan Kinerja Web Server Menggunakan Kubernetes

Raihan Sidik[1], Husaini[2], Ilham Safar[3*]

[1,3] *Jurusan Tekniknologi Informasi dan Komputer Politeknik Negeri Lhokseumawe*
Jln. B.Aceh Medan Km.280 Buketrata 24301 INDONESIA
[1]raihansidik21@gmail.com, [2]husaini@pnl.ac.id, [3]ilham_safar@pnl.ac.id

Abstrak— ***Web server* bertugas untuk melayani *request user* dan memberikan *respon* kepada pengguna setiap waktu. *Web server* memerlukan penambahan layanan tanpa mengganggu proses yang sedang beroperasi agar *server* tetap berfungsi dengan baik. penelitian ini bertujuan untuk dapat meningkatkan kinerja *web server* dalam melayani *request user* agar lebih optimal. Melayani request user dengan lebih optimal dapat dilakukan dengan Skabilitas. Skalabilitas adalah penambahan layanan pada *server* namun tidak mempengaruhi kinerja *server*, hal tersebut dapat dilakukan dengan menerapkan *Autoscaling*. *Autoscaling* digunakan untuk meningkatkan optimasi penggunaan *resource* pada suatu *web server* dalam mempertahankan kinerja yang stabil. Untuk menerapkan Autoscaling dapat digunakan sebuah *platform* bernama Kubernetes. Hasil dari penelitian ini mencakup lima parameter yaitu *Scale Up, Scale Down, Throughput, Response Time*, dan *CPU Usage* serta mengetahui Daya Tahan dari *web server*. Hasil akhir menunjukkan bahwa *web server* mampu menerima beban *request* hingga 1000000 koneksi dengan 10000 koneksi/detik dimana *web server virtual yang* sudah menggunakan Autoscaling mampu menghemat *resource* lebih baik dibandingkan dengan *web server* yang tidak menggunakan *Autoscaling* dengan perbedaan dari dari parameter *Throughput* adalah 23.72 Kbps, perbedaan dari parameter *response time* adalah 5,16 request/detik, dan perbedaan dari parameter *CPU Usage* adalah 5,72 % dimana rata-rata nilai Parameter *Scale Up* adalah 90 detik lalu untuk Parameter *Scale Down* adalah 280 detik dengan rata-rata jumlah *pod* ditambahkan adalah 5,33.**
Kata kunci— Autoscaling,Web Server, Kubernetes.

Abstract— **The web server is in charge of serving user requests and responding to users every time. The web server requires additional services without interrupting the running processes so that the server continues to function properly. This study aims to improve the performance of the web server in serving user requests to make it more optimal. Serving user requests more optimally can be done with scalability. Scalability is the addition of services on the server but does not affect server performance, this can be done by implementing Autoscaling. Autoscaling is used to improve the optimization of resource usage on a web server in order to maintain stable performance. To implement Autoscaling, a platform called Kubernetes can be used. The results of this study include five parameters, namely Scale Up, Scale Down, Throughput, Response Time, and CPU Usage and determine the durability of the web server. The final result shows that the web server is able to accept request loads of up to 1000000 connections with 10000 connections/second where the virtual web server that has used Autoscaling is able to save resources better than the web server that does not use Autoscaling with the difference from the Throughput parameter is 23.72 Kbps, the difference from the Response Time parameter is 5.16 requests/second, and the difference from the CPU Usage parameter is 5.72% where the average value of the Scale Up Parameter is 90 seconds then for the Scale Down parameter is 280 seconds with the average number of pods added is 5.33.**
Keywords - Autoscaling, Webserver, Kubernetes.

I. PENDAHULUAN

Perkembangan teknologi informasi dan komunikasi semakin lama semakin canggih, hal itu dapat diketahui dari kemudahan akses informasi melalui *website*. *Website* merupakan sebuah kumpulan halaman-halaman *web* beserta berkas-berkas pendukungnya, seperti berkas gambar, video, dan berkas digital lainnya yang disimpan pada sebuah *web server* yang umumnya dapat diakses melalui *internet* [1]

Web server merupakan *server* yang berfungsi untuk melayani *request* dan *response* pengguna menggunakan *protokol HTTP* (Hyper Text Transfer Protocol) sebagai *protocol* pengiriman data [2]. Masalah yang sering dialami ketika banyak pengguna mengakses sebuah situs adalah kemampuan *server* dalam menangani permintaan dari pengguna seperti menerima *request* dan mengirim *response HTTP*, hal tersebut akan memerlukan penambahan layanan atau *server* tanpa mengganggu kinerja *server* agar *server* dapat tetap berjalan dengan baik [3].

Scalability atau skalabilitas adalah suatu proses penambahan layanan atau server tanpa mengganggu kinerja *server* [7]. Skabilitas ini dapat dilakukan dengan metode bernama *autoscaling*. Autoscaling akan memonitoring *server* secara berkala ketika sistem mendeteksi layanan pada sebuah *server* tidak dapat memenuhi permintaan pengguna maka sistem akan menambahkan layanan yang akan membantu server menanggapi permintaan pengguna dan jika permintaan pengguna telah menurun layanan yang telah bertambah akan otomatis berkurang [6].

Kubernetes adalah sistem *open source*. *Kubernetes* digunakan untuk mengotomatisasi penyebaran, penskalaan, dan pengelolaan aplikasi kontainer [5]. *Kubernetes* dapat

membantu seorang administrator jaringan untuk melakukan otomasi di servernya, dalam penelitian ini *kubernetes* digunakan untuk mengatur pengelolaan *autoscaling* pada sebuah *web server* agar dapat mengatasi masalah skalabilitas [4]

Penelitian ini diharapkan dapat mempercepat dan mengoptimalkan Kinerja *web server* nantinya dalam menanggapi *request* dari pengguna dan juga dapat menyesuaikan sumber daya yang dimiliki seperti menurunkan atau menambah jumlah layanan sesuai dengan kebutuhan sistem tanpa mengganggu proses yang sedang berjalan

II. METODOLOGI PENELITIAN

A. *Analisis Kebutuhan*

1. *Perangkat Keras*

Perancangan *hardware* yang dibutuhkan yaitu sebuah *PC* dengan spesifikasi yang mencukupi untuk menjalankan beberapa *virtual machine* sekaligus. Spesifikasi *PC* yand direkomendasikan adalah :

a. Minimal *processor 64-bt x86 Intel Core 2 Duo, AMD FX Dual Core.*
b. Minimal *1.3GHz core speed processor (2GHz* direkomendasikan).
c. Minimal *8GB RAM* (16 *GB* direkomendasikan).
d. Minimal *60 GB* Kapasistas *Hardisk* Tersedia (120GB direkomendasikan).
e. Sistem Operasi *PC 64-bit.*

2. *Perangkat Lunak*

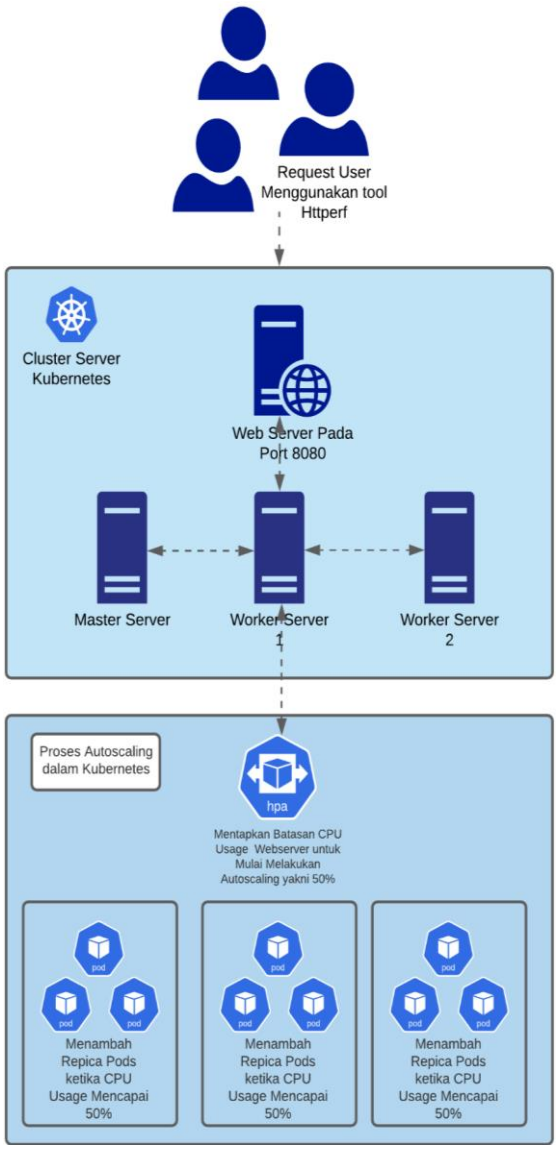
a. *VMware* untuk tempat pembuatan *Virtual Machine*
b. *Kubernetes* untuk Melakukan *Autoscaling*
c. *Httperf* untuk pemberi beban paket *trafik* dan pengujian *server*
d. *Xshell* sebagai *console* untuk *server*

B. *Perancangan Sistem*

1. *Blok Diagram Sistem*

Blok diagram sistem adalah diagram dari sebuah sistem, dimana bagian utama atau fungsi yang diwakili oleh blok dihubungkan dengan garis yang menunjukkan hubungan dari blok. Blok diagram Penelitian Sistem ini secara keseluruhan menjelaskan bagaimana proses penerapan *horizontal pod autoscaler* pada *web server* menggunakan Kubernetes.

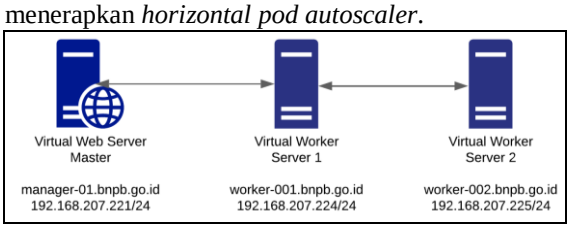
Sistem *autoscaling* yang diterapkan pada *web server* ini dimulai dengan user yang melakukan akses menuju *cluster server kubernetes* yang terdiri dari satu *server master* dan dua *worker server*. *cluster server kubernetes* ini yang akan menerima dan memantau *request* yang didapat dari user nantinya. Ketika *trafik* layanan yang diterima oleh *web server* melebihi batas yang telah ditetapkan pada *horizontal pod autoscaler*, *kubernetes* nantinya akan melakukan *scale up* (Menambahkan Layanan) berupa *pod replika webserver* pada setiap *worker server* hingga mencapai kebutuhan untuk menangani *request* dari *user* atau Setelah mencapai bataas maksimal *replika*. Setelah *request* atau t*rafik* layanan yang diterima *web server* sudah kembali normal, *horizontal pod autoscaler* akan melakukan *scale down* (Mengurangi Layanan) dengan mengurangi *replica pod web server* menjadi seperti semula.



Gambar 1. Diagram Blok Sistem

2. *Perancangan Jaringan Pada Mesin Virtual*

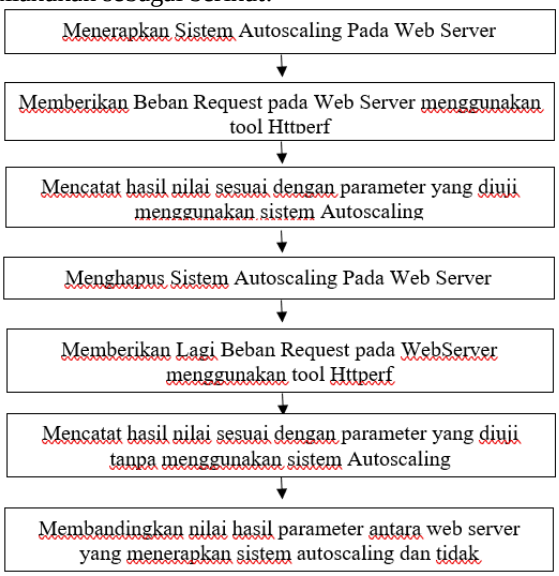
Pada Implementasi *autoscaling* di penelitian ini menggunakan mesin *virtual* yang terdiri dari 1 *master* dan 2 *worker* yang *merupakan* syarat untuk menerapkan *horizontal pod autoscaler*.



Gambar 2. Perancangan Jaringan Pada Mesin Virtual

C. *Tahapan Pengujian Sistem*

Setelah sistem dibuat, dilakukan pengujian untuk memperoleh data yang dibutuhkan. Adapun tahapan dilakukan sebagai berikut:



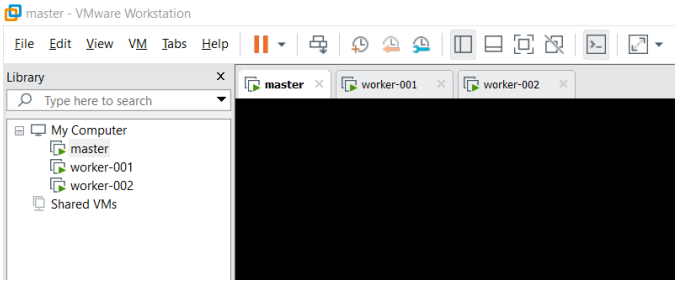
III. HASIL DAN PEMBAHASAN

A. *Implementasi Sistem*

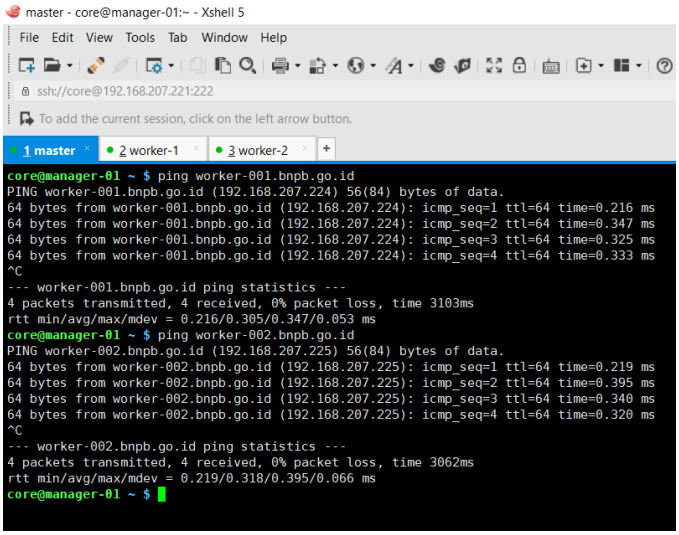
Dalam melakukan konfigurasi *autoscaling* diperlukan minimal 3 *server* yang terdiri dari 1 *master* dan 2 *worker* dan dibangun di *VMware Workstation*. Mesin virtual untuk Master terdiri dengan spesifikasi *RAM 8 GB*, Processor 2 Core, dan *Hardisk 200 GB* sedangkan untuk Worker dengan spesifikasi *RAM* 4 *GB, Processor 2 Core* dan *Memory 400 GB*. Ketiga mesin *virtual* tersebut terhubung satu sama lain dengan jaringan yang sama. Setelah Sistem Operasi di install dan mesin *virtual* berjalan dengan baik.

Pada mesin *virtual* yang akan digunakan harus memiliki *hostname* dan alamat *IP* yang berbeda. Untuk *server master* memiliki *hostname* '*master'* dengan *IP* Address '192.168.207.221/24', untuk *server worker* Satu memiliki *hostname* '*worker1'* dengan *IP Address* '192.168.224/24' dan *server worker* Dua memiliki *hostname* '*worker2'* dengan *IP Address* '192.168.225/24'. Lalu setelah selesai dikonfigurasi dengan benar diperlukan pengujian apakah jaringan berjalan dengan baik, pengujian dilakukan dengan melakukan *ping* dari *master server* kepada kedua *worker server* dengan melakukan *ping* menuju *hostname* dari *worker server*

Dalam *kubernetes* ketiga mesin *virtual* yang telah di instal memiliki peranan yang berbeda, dimana terdapat *master node* dan *worker node*, kedua pernanan tersebut saling terhubung satu sama lain. *Master node* berfungsi sebagai sumber utama aplikasi kontainer dan untuk monitoring serta *controling* sementara *worker* sebagai tempat pembagian beban dan tempat dijalankan dan di prosesnya request yang masuk pada *cluster server*. *Master node* mampu membagi *resource* kepada *worker node* sehingga aplikasi bisa digunakan dengan baik, apabila pada sisi *master node* mengalami server-failure maka aplikasi tetap berjalan sedia kala karena *worker node* mampu mem-*backup* layanan dalam aplikasi tersebut. Inilah yang dimaksud konsep *cluster-server* pada *kubernetes*.

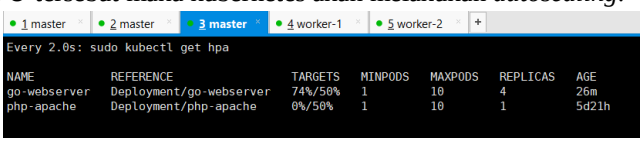


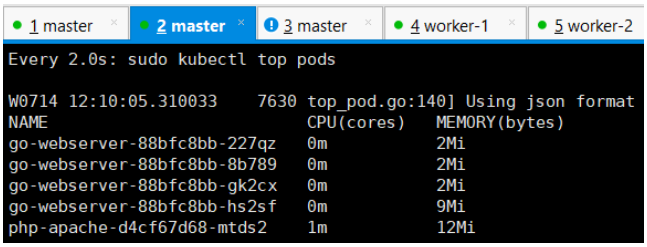
Gambar 3. Tampilan Awal Mesin Virtual



Gambar 4. Pengujian Koneksi Antar Server

Implementasi dijalankan pada *deployment* dengan menjalankan konfigurasi *autoscaling* serta mendeklarasikan sasaran *deployment*, memastikan batasan *CPU* yang dipakai oleh *pod* dan jumlah maksimum *pod* yang diberikan ketika *autoscaling*. dalam konfigurasi *autoscaling* ini memastikan batas maksimum dari *deployment go-webserver*. Setelah itu membuat minumum serta maksimum *pod* dalam makna apabila pada dikala permintaan client tertentu akan membuat *kubernetes* melaksanakan *autoscaling pod* sebanyak jumlah replika yang diperlukan untuk menyeimbangkan jumlah *resource* dengan permintaan client tersebut. Langkah yang terakhir ialah mendefinisikan target *CPU* tiap-tiap pod menjadi 50%, perihal inilah yang jadi kunci Autoscaling. Apabila jumlah permintaan client lebih dari batas sasaran *CPU* tersebut maka kubernetes akan melakukan *autoscaling*.





Gambar 5. Penerapan Horizontal Pod Autoscaling

B. Uji Coba

Parameter yang di uji dalam penelitian kali ini ada 5 yaitu *scale up, scale down, throughput, response time,* dan *CPU usage*.

a. *Scale up* : Menunjukan waktu berapa lama proses penambahan *pod* Ketika terjadi request yang melebihi batas persentasi hingga mencapai batas maksimal
b. *Scale Down* : Menunjukan waktu berapa lama proses pengurangan *pod* (yang ditambahkan ketika request melebihi batas HPA) hingga kembali seperti semula
c. *Throughput* : Menunjukkan banyaknya layanan yang dilayani oleh *server* dalam satu satuan waktu (s).
d. *Response Time*: Mengacu pada seberapa waktu cepat yang dapat dilakukan *server* dalam melayani permintaan client dalam satuan waktu (s).
e. *CPU Usage* : Menunjukkan sumber daya *CPU* yang digunakan pada waktu kinerja *web server* dalam melayani permintaan client.

Berikut skenario pengujian yang akan digunakan dalam penelitian ini :

a. Pengujian untuk mengetahui nilai parameter *scale up* dan *scale down* pada sistem *autoscaling* dimana akan diberikan beban yang terus meningkat mulai dari 10000 koneksi dengan 100 koneksi per detik, dan meningkat secara berkala hingga 50000 koneksi dengan 500 koneksi per detik. Dan nantinya akan dicatat berapa jumlah pods yang ditambahkan untuk mereplika *web server* dari tiap-tiap koneksi yang diberikan, juga akan dicatat waktu yang diperlukan untuk melakukan *scale up* dan *scale down*
b. Pengujian untuk mengetahui nilai *parameter througput, cpu usage, respone time* dengan memberikan beban yang terus meningkat mulai dari 10000 koneksi dengan mulai dari 100 hingga 500 koneksi per detik, dilanjutkan dengan 20000 koneksi mulai dari 100 hingga 500 koneksi perdetik, dilanjutkan dengan 30000 koneksi mulai dari 100 hingga 500 koneksi perdetik, dilanjutkan dengan 40000 koneksi mulai dari 100 hingga 500 koneksi perdetik, dan yang terahir dengan 50000 koneksi mulai dari 100 hingga 500 koneksi perdetik dimana, dimana nantinya akan diuji pada *web server* yang menerapkan sistem autoscaling dan tidak menerapkan sistem autoscaling agar dapat dibandingkan nantinya.
c. Pengujian untuk mengetahui daya tahan server dengan beban secara terus menerus dengan nilai mulai dari 100.000 koneksi dengan 1000 koneksi per detik hingga 1.000.000 koneksi dengan 10.000 koneksi per detik.

Metode analisis yang digunakan untuk membahas hasil pengujian tersebut menggunakan metode deskriptif. Metode deskriptif adalah metode yang digunakan untuk membandingan hasil pengujian *web server* sebelum dan setelah konfigurasi dengan parameter *throughput, response time,* dan *CPU usage*. Adapun kriteria throughput didapatkan dari *variabel NET I/O,* dan kriteria *response time* didapatkan dari variabel *request rate* pada hasil uji coba *httperf*. Sedangkan *CPU usage* dihasilkan dari rata – rata pemantauan pada *node kubernetes.*

Tool yang digunakan pada pnegujian kali ini untuk memberikan *load-generator* adalah Httperf, dimana httperf akan di *deploy* kedalam *kubernetes* berupa *kubernetes Job* yang telah dikonfigurasi di dalam *file yaml*. Isi dari *file yaml* yang akan *dideploy* untuk memberikan *Load-Generator* adalah :

```
apiVersion: batch/v1
kind: Job
metadata:
  name: httperf-bench
spec:
  parallelism: 10
  template:
    spec:
      containers:
      - name: httperf-bench
        image: cyrilbkr/httperf
        args:
        - /bin/sh
        - -c
        - httperf --server mytestserver.com --uri="/" --port 443  --num-conns 1000000 --ra 100
        --timeout 1
        resources:
          requests:
            memory: "64Mi"
            cpu: "1"
          limits:
            memory: "2Gi"
            cpu: "2"
        securityContext:
          allowPrivilegeEscalation: true
      restartPolicy: Never
```

Gambar 6. File Tool Httperf

1) Pengujian Parameter Scale Up

Hasil pengujian performa *web server* terhadap parameter *scale up* ditunjukkan pada Tabel 1.

TABEL 1
PENGUJIAN PARAMETER SCALE UP

No	Jumlah Koneksi	Koneksi/ Detik	Waktu Scale Up	Pod Ditambahkan
1	10000	100	90	3
2	20000	200	90	5
3	30000	300	90	7
4	40000	400	90	9
5	50000	500	90	9
		Rata-rata	90	5,33

Tabel 1 menunjukan bahwa waktu yang diperlukan untuk scale up tetap sama hingga mencapai maksimum *pod* yang diperlukan yaitu 90 detik. dan untuk *Pod* yang ditambahkan semakin meningkat seiring dengan jumlah koneksi yang terus meningkat dari 10000 hingga 50000 dengan rata rata 5,33.

2) Pengujian Parameter Scale Down

Hasil pengujian performa web server terhadap parameter Scale Down ditunjukkan pada Tabel 2.

TABEL 2
PENGUJIAN PARAMETER SCALE DOWN

No	Jumlah Koneksi	Koneksi/ Detik	Waktu Scale Down	Pod Ditambahkan
1	10000	100	300	3
2	20000	200	330	5
3	30000	300	330	7
4	40000	400	360	9
5	50000	500	360	9
		Rata-rata	280	5,33

Tabel 2 Menunjukan bahwa setelah melakukan *scale up* waktu rata rata yang diperlukan untuk Mengurangi layanan / *Pod* adalah 280 detik dimana 300 detik untuk 3 pod, 330 detik untuk 5-6 *pod* dan 360 detik untuk 9 *Pod*

3) Pengujian Parameter *Throughput*

Hasil pengujian performa *web server* terhadap parameter *throughput* ditunjukkan pada Tabel 3.

TABEL 3
PENGUJIAN PARAMETER THROUGHPUT

No	Jumlah Koneksi	Koneksi/ Detik	Tanpa HPA Throughput (Kb/s)	Dengan HPA Throughput (Kb/s)
	10000	100	18.1	18.1
	10000	200	34.1	35.9
1	10000	300	9.9	54.2
	10000	400	11.8	72.3
	10000	500	23.5	90.3
	20000	100	18.1	18.1
	20000	200	36.1	25.2
2	20000	300	17.9	32.9
	20000	400	17.8	46.2
	20000	500	19.0	62.9
	30000	100	18.1	18.1
	30000	200	36.1	36
3	30000	300	17	54.2
	30000	400	17.1	46.2
	30000	500	16.8	40.6
	40000	100	18.1	18.1
	40000	200	35.1	36.1
4	40000	300	16	53.9
	40000	400	16	72.3
	40000	500	16.6	52.8
	50000	100	18.1	18.1
	50000	200	36.1	36.1
5	50000	300	15.9	54.2
	50000	400	15.9	53.6
	50000	500	16.1	52.9
		Rata-rata	23.6	47.32

Tabel 3 menampilan perbandingan nilai parameter *throughput* yang dihasilkan oleh *server* saat sebelum menerapkan autoscaling dan *server* setelah menerapkan autoscaling dimana pengujian dilakukan dengan menunggu *scale down* hingga *server* menjadi seperti semula. *Server* sebelum menerapkan autoscaling menciptakan nilai throughput sebesar 23,6KB/s, sedangkan *server* sesudah menerapkan autoscaling menghasilkan *nilai throughput* sebesar 47,32KB/s. Selisih dari kedua hasil t*hroughput* tersebut sebesar 23.72KB/s.

4) Pengujian Parameter Response Time

Hasil pengujian performa *web server* terhadap parameter *response time* ditunjukkan pada Tabel 4.

TABEL 4
PENGUJIAN PARAMETER RESPONSE TIME

No	Jumlah Koneksi	Koneksi/ Detik	Tanpa HPA Response Time (Requests/s)	Dengan HPA Response Time (Requests/s)
	10000	100	100	100
	10000	200	200	200
1	10000	300	300	300
	10000	400	400	400
	10000	500	500	495.3
	20000	100	100	100
	20000	200	200	198.3
2	20000	300	300	291.6
	20000	400	450	304.5
	20000	500	278.3	352
	30000	100	100	100
	30000	200	200	200
3	30000	300	300	300
	30000	400	250.1	271.4
	30000	500	249	224
	40000	100	100	100
	40000	200	200	200
4	40000	300	300	298.6
	40000	400	250.1	399.9
	40000	500	249	244.4
	50000	100	100	100
	50000	200	200	200
5	50000	300	300	300
	50000	400	336	321.1
	50000	500	294.1	242.9
		Rata-rata	256.72	251.56

Tabel 4 menampilan perbandingan nilai parameter *response time* pada saat pengujian terhadap *server* yang belum menerapkan *autoscaling* sebesar 256,72 *requests*/detik, sedangkan pada *server* yang telah menerapkan autoscaling mendapatkan hasil parameter *response time* sebesar 251,56 *requests*/detik. Hasil uji coba tersebut mendapatkan hasil bahwa *server* sesudah menerapkan *autoscaling* menghasilkan nilai response time yang lebih besar daripada *server* sebelum menerapkan *autoscaling* dengan selisih nilai sebesar 5,16 *requests*/detik

5) Pengujian Parameter *CPU Usage*

Hasil pengujian performa web server terhadap parameter Response Time ditunjukkan pada Tabel 5.

TABEL 5
PENGUJIAN PARAMETER CPU USAGE

No	Jumlah Koneksi	Koneksi/ Detik	Dengan HPA CPU Usage (%)	Tanpa HPA CPU Usage (%)
	10000	100	55%	58%
	10000	200	55%	60%
1	10000	300	56%	62%
	10000	400	52%	64%
	10000	500	52%	60%
	20000	100	56%	59%
	20000	200	56%	61%
2	20000	300	57%	62%
	20000	400	55%	64%
	20000	500	53%	62%
	30000	100	56%	58%
	30000	200	56%	60%
3	30000	300	58%	62%
	30000	400	57%	64%
	30000	500	55%	62%
	40000	100	55%	59%
	40000	200	56%	60%
4	40000	300	54%	62%
	40000	400	55%	60%
	40000	500	53%	62%
	50000	100	56%	58%
	50000	200	56%	60%
5	50000	300	54%	62%

50000	400	56%	60%
50000	500	54%	60%
	Rata-rata	55,12%	60.84%

Tabel 5 menujukkan hasil pengujian *performa web server* mendapatkan hasil parameter *CPU usage* pada *server* stelah menerapkan autoscaling dengan konsumsi *CPU* sebesar 55, 12%, sedangkan konsumsi *CPU* yang dihasilkan pada server sebelum menerapkan autoscaling sebesar 60,84%. Nilai penggunaan *CPU* didapatkan dari nilai hasil pemantauan dari *cluster server* menggunakan perintah "watch sudo kubectl top nodes" dimana dengan perintah watch nodes yang dipantau akan terus di refresh setiap 2 detik sehingga kita dapat melihat jumlah penggunaan *CPU usage* yang terus berubah, dan hasilnya diketetahui konsumsi *CPU server* sesudah menerapkan autoscaling lebih sedikit daripada *server* sebelum menerapkan *autoscaling* dengan selisih 5,72 %.

6) Pengujian Daya Tahan *Web Server*

Hasil pengujian performa *web server* terhadap daya tahan *web server* ditunjukkan pada Tabel 6.

TABEL 6
PENGUJIAN DAYA TAHAN WEB SERVER

No	Jumlah Koneksi	Koneksi/ Detik	Analisis Hasil
1	100000	1000	Koneksi Berhasil, Web Server Mampu Menerima Koneksi yang masuk dengan request rate 255.0 req/s (3.9 ms/req), Autoscaling Berjalan Lancar
2	200000	2000	Koneksi Berhasil, Web Server Mampu Menerima Koneksi yang masuk dengan request rate 311.2 req/s (3.2 ms/req), Autoscaling Berjalan Lancar
3	300000	3000	Koneksi Berhasil, Web Server Mampu Menerima Koneksi yang masuk dengan request rate 905.0 req/s (1.1 ms/req), Autoscaling Berjalan Lancar
4	400000	4000	Koneksi Berhasil, Web Server Mampu Menerima Koneksi yang masuk dengan request rate 684.3 req/s (1.5 ms/req), dan Autoscaling Berjalan Lancar
5	500000	5000	Koneksi Berhasil, Web Server Mampu Menerima Koneksi yang masuk dengan request rate 743.0 req/s (1.3 ms/req), dan Autoscaling Berjalan Lancar
6	600000	6000	Koneksi Berhasil, Web Server Mampu Menerima Koneksi yang masuk dengan request rate 973.9 req/s (1.0 ms/req), dan Autoscaling Berjalan Lancar
7	700000	7000	Koneksi Berhasil, Web Server Mampu Menerima Koneksi yang masuk dengan request rate 853.3 req/s (1.2 ms/req), dan Autoscaling Berjalan Lancar
8	800000	8000	Koneksi Berhasil, Web Server Mampu Menerima Koneksi yang masuk dengan request rate 654.2 req/s (1.5 ms/req), dan Autoscaling Berjalan Lancar
9	900000	9000	Koneksi Berhasil, Web Server Mampu Menerima Koneksi yang masuk dengan request rate 703.7 req/s (1.4 ms/req), dan Autoscaling Berjalan Lancar
10	1000000	10000	Koneksi Berhasil, Web Server Mampu Menerima Koneksi yang masuk dengan request rate 640.8 req/s (1.6 ms/req), dan Autoscaling Berjalan Lancar

Tabel 6 menujukkan hasil Bahwa Web Server yang dibuat mampu menahan beban request hingga 1000000 koneksi dengan koneksi 10000 per detik. Sistem Autoscaling mampu berjalan dengan lancer selama periode pemberian beban. Namun yang perlu diperhatikan dibagian request rate dimana Web Server memang dapat menerima setiap request yang masuk namun tidak dapat membalas seluruh request yang masuk semakin banyak beban per detik yang masuk maka semakin sedikit pula yang dapat di respon oleh Web Server.

Setelah dilakukan pengujian terhadap parameter *Scale Up, Scale Down,* Throughput, Respone Time, *CPU Usage* dan Daya tahan server maka dapat diambil hasil bahwa *server* setelah konfigurasi lebih baik daripada *server* sebelum konfigurasi. Hasil pengujian kinerja *web server* terhadap parameter *throughput* pada server sebelum menerapkan *Autoscaling* menghasilkan 23,6 KB/s, sedangkan *server* yang telah menerapkan *Autoscaling* menghasilkan 47,32 *KB*/s. Selisih dari kedua hasil *Throughput* tersebut sebesar 23.*72KB*/s. Besaran nilai *throughput* yang dihasilkan menampilkan bahwa kinerja web server menjadi lebih baik dikarenakan banyaknya permintaan *client* yang mampu dilayani oleh *web server.* Dari hasil pengujian parameter *throughput* yang telah dilakukan, dapat disimpulkan bahwa *server* yang menerapkan *auto scaling* lebih cepat melayani permintaan *client* daripada *server* yang tidak menerapkan *Autoscaling.*

Hasil pengujian kinerja web server terhadap parameter *response time* pada server yang tidak menerapkan *Autoscaling* menciptakan nilai *response time* sebesar 256,72 *request*s/detik. Sedangkan nilai *response time* yang diperoleh *server* yang telah menerapkan *Autoscaling* menciptakan nilai sebesar 251,56 *request*s/detik, dengan selisih diantara keduanya yaitu 5,16 *request*/detik . Jadi nilai *response time* yang dihasilkan oleh *server* sesudah menerapkan *autoscaling* lebih kecil daripada server sebelum menerapkan *autoscaling*. Nilai *response time* mewakili kecepatan *web server* dalam menjawab permintaan dari *client*. Jika nilai *response time* yang dihasilkan kecil maka web server semakin cepat dalam menanggapi permintaan *client*. selisih perbandingan dari hasil pengujian tersebut sebesar 5,16 request/detik.

Hasil pengujian performa server terhadap parameter pemakaian *CPU* (*CPU usage*) pada server yang tidak menerapkan *Autoscaling* sebesar 55,12%, dan server yang menerapkan *Autoscaling* sebesar 60,84%. Selisih perbandingan hasil tersebut hingga 5,72%. Pemakaian *CPU* mewakili kinerja web *server* secara keseluruhan, semakin tinggi pemakaian *CPU* pada *server* berarti semakin lemah pula *performa CPU* yang dihasilkan

Hasil pengujian performa daya tahan *web server* menunjukan bahwa *web server* mampu menerima request hingga 1000000 koneksi dengan koneksi 10000 per detiknya namun *web server* tidak dapat merespon seluruh requestnya dengan baik sekalipun sistem Autoscaling Berjalan dengan lancar. Ini menunjukan bahwa sekalipun *web server* mampu menerima request yang banyak dengan bantuan *autoscaling*

namun tidak 100% *request* tersebut mampu di *respon* oleh *web server.*

IV. KESIMPULAN

Cluster server kubernetes dengan penerapan sistem *horizontal pod autoscaler* dengan *web server* yang berbasis *container* dibangun di atas *VMware* Workstation berhasil diimplementasikan. Pengujian Autoscaling dilakukan dengan memberikan beban menggunakan tool *httperf* yang telah ditentukan Koneksi dan jumlah koneksi per detik nya lalu di *deploy* menjadi Kubernetes *job.* Parameter pengujiannya adalah *scale up, scale down, throughput, response time*, dan *CPU* usage dan mengukur daya tahan dari *webserver*. Dari hasil yang didapatkan menunjukan bahwa *web server* setelah menerapkan autoscaling lebih baik dari pada web server sebelum menerapkan *autoscaling* dengan perbandingan nilai *throughput* 23.72 *KB/*detik, nilai *response time*5,16 *request*s/detik dan Nilai *CPU usage* sebesar 5,72%.

REFERENSI

[1] Adnan, F., & Kusnawi. (2016). Analisis Perbandingan Performa Web Server Apache dan Nginx menggunakan Httperf pada VPS dengan Sistem Operasi CentOs. *Stmik Amikom Yogyakarta*, *3*(1), 6.

[2] Arman, M. (2016). Analisa Kinerja Web Server E-learning Menggunakan Apache Benchmark dan Httperf. *Jurnal Integrasi*, *8*(2), 93–100.

[3] Jusuf H. (2016). Penggunaan Secure Shell (SSH) Sebagai Sistem Komunikasi Aman Pada Web Ujian Online. *Bina Insani Ict Journal*, *2*, 75–84.

[4] Rosyadi, I., Utama, S. N., & Putra, O. V. (2019). Implementation *Autoscaling* Container Web Server using Kubernetes Promox-Based on Server University of Darussalam Gontor Implementation *Autoscaling* Container Web Server using Kubernetes Promox-Based on Server University of Darussalam Gontor. *Jurnal Rekayasa Sistem Dan Industri*, *June*. https://doi.org/10.25124/jrsi.v6i02.362.

[5] Sumbogo, Y. T., Data, M., & Siregar, R. A. (2018). Implementasi Failover Dan *Autoscaling* Kontainer Web Server Nginx Pada Docker Menggunakan Kubernetes. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer (J-PTIIK) Universitas Brawijaya*, *2*(12), 6849–6854.

[6] Widarma, A., & Siregar, Y. H. (2019). Analisis Kinerja Teknologi Virtualisasi Server (Study Kasus : Universitas Asahan). Vm, 688–698.

[7] Zakaria. (2017). Analisa Performansi Server Cloud Berbasis Proxmox Ve untuk Multi Server dan Multi Platform pada Praktikum Administasi Jaringan Komputer. 2(1), 17–26.