ANALISIS PENJAMINAN AVAILIBILITY APLIKASI BERBABASIS WEB TERHADAP SERANGAN DDOS MENGGUNAKAN AMAZON EKS

Gilang Adhi Wibowo[1], Siti Amatullah Karimah[2], Febri Dawani[3]
[1,2,3] Universitas Telkom
glngadhi@student.telkomuniversity.ac.id[1], karimahsiti@telkomuniversity.ac.id[2], febridawani@telkomuniversity.ac.id[3]

Abstrak

Ketersedian dari sebuah sistem dalam menerima serangan DdoS HTTP *Flood* menjadi permasalahan dalam penelitian ini. Serangan *Denial of Service* (DoS) dan *Distributed Denial of Service* (DDoS) menjadi ancaman utama terhadap industri TI dan jaringan komputer saat ini. Serangan semacam ini bertujuan untuk membuat sumber daya jaringan atau sistem tidak tersedia bagi pengguna sehingga tidak ada orang yang dapat mengaksesnya. Pendirian sistem infrastruktur perlu kemampuan *scalability* secara otomatis agar bisa menerima serangan DdoS. *Platform* yang dapat digunakan untuk mendukung ketersediaan tinggi sebuah aplikasi adalah *container orchestrator* (Kubernetes). Penelitian ini berfokus pada perancangan *cloud computing* sehingga mampu menerima serangan HTTP *Flood* sebanyak 5.000, 10.000, 15.000, 30.000 yang dilakukan masing-masing 10 kali. Hasil penelitian ini menunjukkan bahwa walaupun terjadi peningkatan jumlah *node/worker* dari 2 menjadi 3, serta adanya peningkatan CPU dan memori yang cukup signifikan, sistem yang dibangun berhasil menangani puluhan ribu serangan. Sehingga, dapat dikatakan bahwa sistem yang dibangun selalu *available* dan dapat diandalkan untuk produksi di dunia kerja.

Kata kunci-DDoS, , orchestrator. ketersediaan

I. PENDAHULUAN

Tidak tersedianya sebuah sistem yang disebabkan oleh Distributed Denial of Service (DDoS) menjadi ancaman utama bagi sebuah industry TI yang berjalan [1]. DDoS adalah serangan berbahaya yang dapat menghabiskan sumber daya server (Bandwidth, Memori, CPU, dll.) dan dapat menyebabkan sebuah server mati dan sebuah sistem tidak tesedia bagi user [5], dengan tidak tersedianya sebuah layanan bagi pengguna tentu saja bisa membuat bisnis TI tidak berjalan dan menimbulkan kerugian. Masalah ketersediaan server dan aplikasi sangat penting bagi pengguna. Ketersediaan server sangat berhubungan erat dengan kualitas yang ditawarkan oleh sebuah aplikasi, karena ketika sebuah server dan aplikasi tidak dapat melayani suatu permintaan, maka layanan tersebut dapat dikatakan kurang baik, ditambah lagi, ketika sebuah server atau sebuah aplikasi mati, layanan secara otomatis akan berhenti dan perlu dilakukan tindakan manual bagi tim operaion untuk menyalakannya kembali. Salah satu platform yang dapat digunakan untuk mendukung ketersediaan tinggi sebuah server dan aplikasi adalah Kubernetes[4]. Containerization adalah proses membundel sebuah aplikasi bersama dengan semua file konfigurasi, library, dan dependensi yang diperlukan agar aplikasi dapat berjalan secara efisien, sebuah bandel tersebut disebut container [6]. Sebuah container adalah sebuah wadah tertutup untuk perangkat lunak yang di dalamnya memiliki semua files dan libraries yang dibutuhkan untuk menjalankan aplikasi [7], teknologi container yang populer saat ini adalah Docker [6]. Kubernetes adalah sistem manajemen container atau yang biasa disebut container orchestrator yang bisa melakukan deployment aplikasi di beberapa server. kubernetes menggunakan teknologi docker untuk menjalankan, melakukan penskalaan container di dalam cluster. Kubernetes dapat membantu menghindari masalah seperti server downtime, kerusakan fisik, kegagalan sistem operasi, dll. Kubernetes dapat memastikan keandalan dan ketersediaan sebuah aplikasi [4].

II. KAJIAN TEORI

Dalam penelitian [1] sudah dilakukan pengujian dan analisis DoS/DdoS menggunakan http flood di atas container orchestration Docker Swarm, CPU Usage mengalami kenaikan yang cukup signifikan dari mulai 0 sampai 44 % saat serangan terjadi, response time juga menaiki kenaikan sebesar 16000 ms atau 16s dari bermula time response adalah 0s. Memory usage juga menaiki kenaikan 0.5 MB dari permulaan 1.5 MB hingga 2MB. Hal ini menyimpulkan bahwa sebuah aplikasi dan infrastruktur bisa kewalahan dalam menangani ribuan atau bahkan jutaan permintaan dari user.

Dalam penelitian [2] juga sudah dilakukan serangan DoS/DDoS dari *mobile device* menggunakan tools yang sama LOIC RSS dengan penelitian [1] dan ada tambahan beberapa tools yang digunakan LOIC dengan metode IRC dan slowbot, didapati hasil bahwa bandwidth tools LOIC memiliki 1000000Bytes atau 1MB, berbeda dengan slowbot tools yang hanya menggunakan 100000 Bytes atau 0.01MB. didapati juga memory client yang menggunakan slowBot *client* memiliki kenaikan yang cukup signifikan yaitu 24MB

pada 500 *connections.*

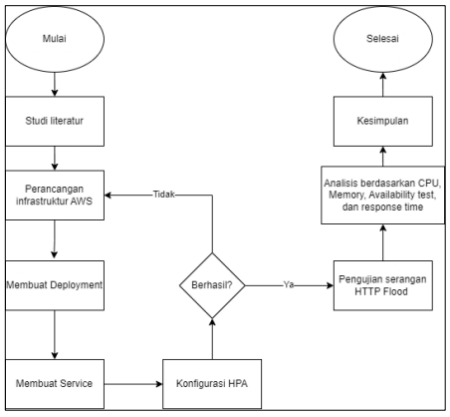
A. Availibility

Availability adalah sebuah konsep dimana sumber daya server terus tersedia walaupun terjadinya banyak gangguan ataupun serangan[4]. Salah satu metode untuk tercapainya availability adalah dengan menggunakan sistem cluster atau Kubernetes[4]. Keuntungan dari sistem cluster atau Kubernetes ini adalah memungkinkan server untuk menyediakan layanan dengan availability tinggi. Ketika satu node/server memiliki masalah, master node akan mendeteksi kegagalan dan mengkonfigurasi ulang sistem sehingga layanan pada nod/server yang bermasalah akan diambil alih oleh node/server lainnya sehingga selalu terciptanya availability [8].sebuah availability dapat diliat dengan cara menghitungnya menggunakan rumus berikut[8]:

$$Availability = \frac{MTBF}{(MTBF + MTTR)} x100 \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

B. *Flowchart* Diagram Sistem

Gambar 1 menunjukan alur pembangunan sistem dalam penelitian ini



GAMBAR 1
METODOLOGI PENELITIAN

C. *Horizontal Pod AutoScaler*

Di Kubernetes, Horizontal Pod Autoscaler secara otomatis memperbarui sumber daya beban kerja (seperti *Deployment* atau *StatefulSet*), dengan tujuan menskalakan beban kerja secara otomatis agar sesuai dengan permintaan. Penskalaan horizontal berarti bahwa respons terhadap peningkatan beban adalah dengan men-*deploy* lebih banyak Pod. Ini berbeda dengan penskalaan vertikal, yang bagi Kubernetes berarti menetapkan lebih banyak sumber daya memori atau CPU ke Pod yang sudah berjalan untuk beban kerja. Berdasarkan penelitian[15] menetapkan ambang batas pemanfaatan atas sumber daya CPU hingga sekitar 50% – 75% menguntungkan aplikasi web. Batas pod untuk melakukan penskalaan otomatis di nilai 75% CPU *utilization* masing-masing pod sesuai dengan yang tercantum pada [15].

```
skripsi > personal-website > hpa-personal-website.yaml
You, 7 days ago | 1 author (You)
1  apiVersion: autoscaling/v2        You, last
2  kind: HorizontalPodAutoscaler
3  metadata:
4    name: hpa-personal-website
5  spec:
6    scaleTargetRef:
7      apiVersion: apps/v1
8      kind: Deployment
9      name: personal-website
10   minReplicas: 1
11   maxReplicas: 10
12   metrics:
13     - type: Resource
14       resource:
15         name: cpu
16         target:
17           type: Utilization
18           averageUtilization: 75
```

GAMBAR 2
KONFIGURASI *HORIZONTAL POD AUTOSCALER*

D. *Deployment*

Deployment adalah konstruksi Kubernetes yang mengontrol pembuatan dan penghancuran pod. Konstruksi ini sangat penting karena itulah yang membuat aplikasi tetap hidup. *Deployment* pada dasarnya adalah kontrak yang dibuat dengan Kubernetes yang menyatakan kondisi berjalannya aplikasi. *Deployment* mendeskripsikan sebuah *state* yang diinginkan dalam berjalannya aplikasi.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: personal-website
  labels:
    app: personal-website
spec:
  selector:
    matchLabels:
      app: personal-website
  template:
    metadata:
      labels:
        app: personal-website
    spec:
      containers:
      - name: personal-website
        image: gilangadhiw/personal-website:1.0
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: 64Mi
            cpu: 10m
          limits:
            memory: 128Mi
            cpu: 30m
```

GAMBAR 1
KONFIGURASI *DEPLOYMENT.*

E. *Service*

Deployment bertugas untuk memastikan bahwa aplikasi berjalan di dalam *cluster*, tetapi kita tidak memiliki cara untuk mengaksesnya dari luar. Di sinilah *Service* berguna. Sebuah layanan digunakan untuk mengatur bagaimana lalu lintas jaringan diteruskan ke pod yang berjalan di suatu tempat dalam sebuah *cluster*.

GAMBAR 4
KONFIGURASI *SERVICE*

F. Kebutuhan Sistem.

Tabel 1 menunjukan kebuthan perangkat keras

TABEL 1
KEBUTUHAN SISTEM

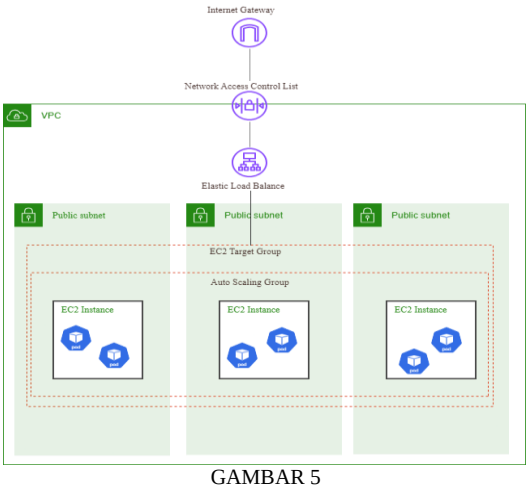
Komponen	Spesifications
Tipe instance	t3.medium
RAM	4 GiB memory
CPU	2vCPUS (2.5 Ghz)
HDD	50 GB HDD

Perangkat lunak yang digunakan:

1. Grafana untuk monitoring Kubernetes dan Docker.
2. Metrics Servers.
3. Kubernetes sebagai *container controller*.
4. Docker sebagai container *virtualization*.
5. Low Orbit Ion Cannon (LOIC).
6. Visual Studio Code sebagai *code editor*.

G. Perancangan Infrastruktur Kubernetes pada Amazon *Web Services.*

Rancangan 3 *Availability Zone* (AZ) dengan 1 *instances* secara minimum di masing-masing AZ untuk meminimalkan kegagalan dalam satu AZ, masing-masing EC2 *Instances* berada dalam Auto *Scaling Group* yang sama untuk mempermudah pembuatan aturan dalam penskalaan EC2 *Instance* tersebut. EC2 *Instances* juga terdapat *security group* yang bertindak sebagai *firewall* untuk mengatur lalu lintas jaringan yang di perbolehkan masuk dan keluar dari Kubernetes *cluster* itu sendiri, kemudian membuat *Elastic Load Balance* (ELB) untuk mengatur beban jaringan atau *traffic* ke dalam masing-masing instances.



GAMBAR 5
TOPOLOGI INFRASTRUKTUR AWS.

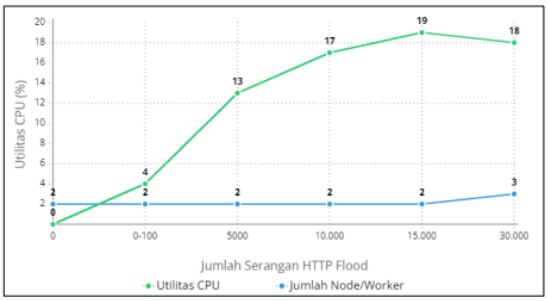
III. METODOLOGI

Untuk mencapai tujuan dari penelitian ini metode yang digunakan adalah berfokus pada perancangan kuberenetes pada layanan *cloud computing.*

IV. HASIL DAN PEMBAHASAN

A. Skenario 1

Pada skenario pertama, dilakukan pengujian serangan yang dilakukan sebanyak 5.000, 10.000, 15.000, 30.000 HTTP *Flood* yang masing-masing dilakukan 10 kali, lalu dilakukannya analisa sistem yang didirikan di atas *container orchestrator* (Kubernetes) dengan mengamati penggunaan CPU. Hasil rata-rata dan perbandingan penggunaan CPU dari jumlah serangan tersebut menunjukan bahwa terjadi kenaikan seiring dengan meningkatnya jumlah serangan. Gambar 6 menunjukan grafik detail hasil pengujian skenario 1.

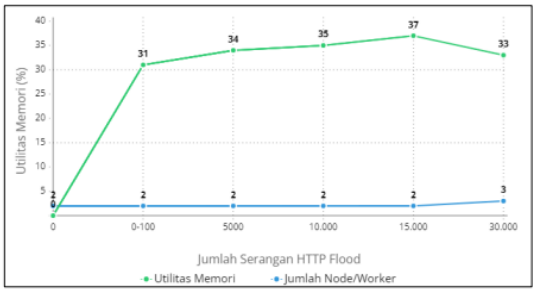


GAMBAR 6
GRAFIK UTILITAS CPU

B. Skenario 2

Pada skenario kedua, dilakukan pengujian dengan jumlah serangan sama, yaitu 5.000, 10.000, 15.000, 30.000 HTTP Flood yang masing-masing dilakukan 10 kali, lalu dilakukannya analisa sistem yang didirikan di *atas container orchestrator* (Kubernetes) dengan mengamati penggunaan memori. Hasil rata-rata dan perbandingan penggunaan memori dari jumlah serangan tersebut menunjukan bahwa terjadi kenaikan seiring dengan meningkatnya jumlah

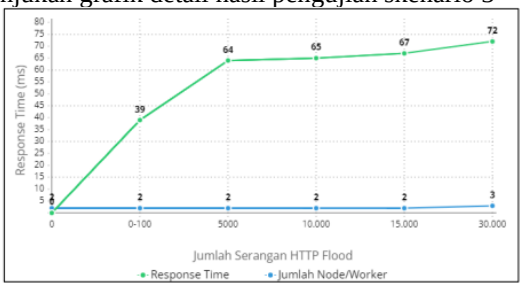
serangan. Gambar 7 menunjukan grafik detail hasil pengujian skenario 2.



GAMBAR 7.
GRAFIK UTILITAS MEMORI

C. Skenario 3

Pada skenario ketiga, dilakukan pengujian dengan jumlah sama, yaitu 5.000, 10.000, 15.000, 30.000 permintaan HTTP *Flood* yang masing-masing dilakukan 10 kali, lalu dilakukannya analisa *response time* yang diberikan oleh web aplikasi menggunakan httping [1]. Gambar 8. Menunjukan grafik detail hasil pengujian skenario 3



GAMBAR 8.
GRAFIK *RESPONSE TIME*

D. Skenario 4

Pada skenario keempat, dilakukan pengujian serangan dengan jumlah sama, yaitu 5.000, 10.000, 15.000, 30.000 permintaan HTTP *Flood* yang masing-masing dilakukan 10 kali, lalu dilakukannya analisa *availability testing* setelah serangan terjadi untuk melihat apakah ada gangguan terkait ketersediaan web aplikasi tersebut menggunakan rumus [1], Tabel 2 menunjukan hasil perhitungan dan pengujian

TABEL 2
.*TABLE AVAILABILITY*

Date of node Down	Date of worker/node Down	MTBF (Minutes)	MTTR (Minutes)	Availability (%)
Never Down	Never Down	38606.4	0 minutes	100
Never Down	Never Down	41126.4	0 minutes	100

Analisa dari hasil pengujian adalah sebagai berikut:

1. Dari pengujian pertama dengan membanjiri serangan sebanyak 5.000, 10.000, 15.000, 30.000 HTTP *Flood* sebanyak masing-masing 10 kali menggunakan *tools* LOIC. Sebelum serangan dilakukan utilitas CPU sebesar 4%, hingga pada 15.000 serangan mencapai 19% dengan jumlah node/*worker* sebanyak 2, namun ketika serangan meningkat menjadi 30.000 utilitas CPU menurun menjadi 18% hal ini disebabkan oleh adanya peningkatan jumlah node/*worker* menjadi 3.
2. Dari pengujian kedua dengan membanjiri serangan sebanyak 5.000, 10.000, 15.000, 30.000 HTTP Flood sebanyak masing-masing 10 kali menggunakan *tools* LOIC. Sebelum serangan dilakukan utilitas memori sebesar 31%, hingga pada 15.000 serangan mencapai 37% dengan jumlah node/*worker* sebanyak 2, namun ketika serangan meningkat menjadi 30.000 utilitas memori menurun menjadi 33% hal ini disebabkan oleh adanya peningkatan jumlah node/worker menjadi 3.
3. Dari pengujian ketiga dengan membanjiri serangan sebanyak 5.000, 10.000, 15.000, 30.000 HTTP *Flood* sebanyak masing-masing 10 kali menggunakan tools LOIC. Sebelum serangan dilakukan *response time* sebesar 39 ms, hingga pada 15.000 serangan mencapai 67 ms dengan jumlah node/*worker* sebanyak 2, namun ketika serangan meningkat menjadi 30.000 *response time* juga mengalami kenaikan menjadi 72 ms hal ini disebabkan oleh adanya peningkatan jumlah node/*worker* menjadi 3.
4. Dari pengujian keempat, didapati *uptime* node/*worker* tidak mengalami perubahan menjadi 0 setelah serangan DDoS menggunakan HTTP *Flood* diluncurkan. Hal ini membuktikan bahwa hipotesa dan tujuan awal dari peneliain ini dapat dibuktikan.

V. KESIMPULAN

Kesimpulan dari hasil pengujian dan analisis pada penelitian ini adalah sistem yang dibangun berhasil menangani puluhan ribu serangan sehingga menciptakan aplikasi dan infrastruktur yang selalu *available* dan dapat diandalkan untuk produksi di dunia kerja. Meskipun terjadi kenaikan CPU, Memori dan *Response Time* yang cukup signifikan dengan penambahan serangan dan dilakukan sebanyak 40 kali namun ketersediaan web aplikasi yang diuji dapat tetap tersedia dibuktikan dengan *availability* test yang sudah disebutkan di poin sebelumnya, hal tersebut dapat dicapai dengan adanya infrastruktur yang memadai sesuai dengan tujuan dari penelitian pada penelitian ini.

REFERENSI

[1] *organizer. Ranganathan Engineering College and Institute of Electrical and Electronics Engineers, Behavioral Analysis of Docker Swarm under DoS/ DDoS Attack*. 2018.

[2] N. Tripathi and B. Mehtre, *DoS and DDoS Attacks: Impact, Analysis and Countermeasures*. 2013.

[3] *organizer. Ranganathan Engineering College and Institute of Electrical and Electronics Engineers, Proceedings of the International Conference on Inventive Communication and Computational Technologies* : ICICCT 2018 : 20-21, April 2018.

[4] A. A. Khatami, Y. Purwanto, and M. F. Ruriawan, "*High availability storage server with kubernetes," in 2020 International Conference on Information Technology*

Systems and Innovation, ICITSI 2020 - Proceedings, Institute of Electrical and Electronics Engineers Inc., Oct. 2020, pp. 74–78. doi: 10.1109/ICITSI50517.2020.9264928.
[5] *Institute of Electrical and Electronics Engineers and IEEE Communications Society, Comparative Study of Security Methods against DDOS Attacks in Cloud Computing Environment.*
[6] Shri Sant Gajanan Maharaj College of Engineering, *Institute Of Electrical And Electronics Engineers. Bombay Section, And Institute Of Electrical And Electronics Engineers, Self-Hosted Kubernetes: Deploying Docker Containers Locally With Minikube*.
[7] S. Chakrabarti et al., *Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform.*
[8] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari, *"A Scalable and Highly Available Web Server*."
[9] S. R. Rizvi, A. Lubawy, J. Rattz, A. Cherry, B. Killough, and S. Gowda, "*A Novel Architecture of Jupyterhub on Amazon Elastic Kubernetes Service for Open Data Cube Sandbox," in International Geoscience and Remote Sensing Symposium (IGARSS), Institute of Electrical and Electronics Engineers Inc*., Sep. 2020, pp. 3387–3390. doi: 10.1109/IGARSS39084.2020.9323748.
[10] M. S. Islam Shamim, F. Ahamed Bhuiyan, and A. Rahman, "*XI Commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices," in Proceedings* - 2020 IEEE *Secure Development, SecDev 2020, Institute of Electrical and Electronics Engineers Inc*., Sep. 2020, pp. 58–64. doi: 10.1109/SecDev45635.2020.00025.
[11] *Computing Conference 2017 London, Institute of Electrical and Electronics Engineers, SAI Computing Conference* 2017.07.18-20 London, *Computing Conference* 2017.07.18-20 London, and SAI 2017.07.18-20 London, *Comparison of the Cloud Computing Platforms Provided by Amazon and Google*.
[12] Han'guk T'ongsin Hakhoe, IEEE *Communications Society, Denshi Jōhō Tsūshin Gakkai (Japan). Tsūshin Sosaieti, and Institute of Electrical and Electronics Engineers, Predictive Container Auto-Scaling for Cloud-Native Applications.*
[13] S. Kho Lin et al., "*Auto-Scaling a Defence Application across the Cloud Using Docker and Kubernetes," in Proceedings* - 11th IEEE/ACM *International Conference on Utility and Cloud Computing Companion, UCC Companion 2018, Institute of Electrical and Electronics Engineers* Inc., Jan. 2019, pp. 327–334. doi: 10.1109/UCCCompanion.2018.00076.
[14] S. Secci, IEEE *Communications Society, International Federation for Information Processing, and Institute of Electrical and Electronics Engineers, Effective Analysis of Secure Web Response Time.*
[15] X. Liu et al., *Auto Scaling Strategy for Amazon Web Services in Cloud Computing.*