JEPIN
(Jurnal Edukasi dan Penelitian Informatika)
ISSN(e): 2548-9364 / ISSN(p) : 2460-0741

Vol. 9
No. 2
Agustus
2023

Honeypot-as-a-*Service* dengan Kubernetes *Cluster*

Rahmat Purwoko[#1], Dimas Febriyan Priambodo [#2], Ghiffari Adhe Permana[#3], Wawan Laksito Yuly Saptomo[*4], Sri Siswanti[*5], Muhammad Hasbi[*6]

[#]*Rekayasa Keamanan Siber, Politeknik Sibar dan Sandi Negara*
Jl. H. Usa, Ciseeng, Bogor 19120
[1]rahmat.purwoko@poltekssn.ac.id
[2]dimas.febriyan@ poltekssn.ac.id
[3]ghiffari.adhe@poltekssn.ac.id

[*]*STMIK Sinar Nusantara*
Jl. K.H Samanhudi No.84-86, Purwosari, Laweyan, Surakarta, Jawa Tengah 57149
[4]wlaksito@sinus.ac.id
[5]syswanty@gmail.com
[6]mhasbi@sinus.ac.id

Abstrak— ***Honeypot* merupakan salah satu strategi yang digunakan untuk melindungi jaringan dari serangan siber. *Honeypot* digunakan untuk menarik penyerang agar menyerang *honeypot* tersebut daripada perangkat-perangkat jaringan. Namun, penggunaan *honeypot* masih jarang terjadi di tingkat korporat maupun individu karena membutuhkan tenaga profesional dan infrastruktur khusus untuk mengelolanya selama implementasi. Berdasarkan permasalahan ini, peneliti mengusulkan solusi *Software*-as-a-*Service* (SaaS) yang disebut Honeypot-as-a-*Service* (HaaS). HaaS adalah layanan *honeypot* berbasis *cloud* yang dikelola oleh orkestrasi kontainer Kubernetes *Cluster*. Penggunaan Kubernetes *Cluster* dirancang untuk mengotomatiskan konstruksi, penjadwalan, pemeliharaan, dan penghapusan *honeypot* berkontainer. Otomatisasi ini dimaksudkan untuk membantu pelanggan yang ingin menggunakan sistem pertahanan berbasis *honeypot* dalam jaringan mereka tanpa harus menjalankan *honeypot* mereka sendiri. Pengguna dapat mendaftar akun dan mengonfigurasi *honeypot* menggunakan *dashboard* yang langsung terhubung ke *cloud honeypot*. Sistem ini sedang dikembangkan di lingkungan pusat data Departemen Keamanan Siber dari Politeknik Siber dan Sandi Negara, yang dikelola dengan manajemen virtualisasi Proxmox Virtual Environment. Komponen-komponen dari sistem HaaS terdiri dari honeypot di Kubernetes *Cluster*, HaaS-*proxy*, dan HaaS Dashboard. Sistem yang telah dibuat kemudian diuji *availability, performance, functionality, and scenario*. Hasil evaluasi sistem menunjukkan bahwa sistem HaaS membutuhkan pengembangan lebih lanjut. Meskipun ketersediaan dan performa sistem HaaS telah memenuhi kriteria layanan berbasis cloud, namun fungsionalitas sistem tidak memenuhi standar layanan SaaS secara umum. Namun, honeypot dibangun untuk memenuhi tujuan honeypot dalam menarik penyerang.**

***Kata kunci*— *Honeypot, Cloud Service , Software*-as-a-*Service* , Kubernetes.**

I. PENDAHULUAN

Peran *honeypot* dalam mengurangi serangan dan mengumpulkan jejak digital sudah sangat dikenal secara umum. Penggunaan teknologi *honeypot* telah terbukti efektif untuk digunakan, namun implementasinya masih sangat jarang dan kurang dimanfaatkan, terutama untuk perusahaan atau organisasi kecil dan kelas menengah, dikarenakan melakukan konfigurasi pada *honeypot* dan mengimplementasikannya dalam sebuah perusahaan atau organisasi membutuhkan tenaga ahli dan infrastruktur yang beragam [1].

Berdasarkan studi yang telah dilakukan oleh University of Maryland, diperkirakan terdapat percobaan serangan setiap 39 detik dengan rata-rata serangan sebanyak sebanyak 2.244 kali dalam hitungan hari [2]. Melihat begitu banyaknya percobaan serangan terhadap jaringan komputer yang semakin lama semakin bervariasi dalam hal metode yang digunakan, mengaplikasikan mekanisme pertahanan keamanan jaringan untuk mengenali berbagai jenis serangan-serangan ini adalah hal yang sangat penting. Implementasi sebuah mekanisme keamanan informasi jaringan satu diantaranya adalah *honeypot*.

Kontainer memberikan kemudahan dalam mengimplementasikan layanan secara cepat, mudah dan murah dalam suatu sistem atau jaringan [3]. Penggunaan kontainer ditujukan untuk mempermudah implementasi sebuah layanan ke dalam sebuah sistem. Penggunaan kontainer secara teori dapat mempermudah implementasi *honeypot* ke dalam sebuah sistem atau jaringan. Pada penelitian Gupta, C. - HoneyKube: Designing a *Honeypot* Using Micro*service* s-Based Architecture [4]. *Honeypot* didesain menggunakan arsitektur micro*service* berbasis Kubernetes *Cluster*. Sebuah *cluster* berisikan *node* layanan dalam bentuk kontainer yang merupakan bagian dari simulasi sistem *honeypot*. Setiap *node* berisikan sebuah

lingkungan *honeypot* yang membentuk sebuah lingkungan individu yang saling berkomunikasi antar *node*.

Berdasarkan pembahasan diatas, penelitian ini menyuguhkan sistem *Honeypot*-as-a*Service* (HaaS) yang menggabungkan kemampuan *honeypot* dengan kemudahan komputasi awan dengan model SaaS. *Honeypot* dikemas ke dalam sebuah image untuk dijalankan pada Kubernetes *Cluster*. Penggunaan Kubernetes *Cluster* memberikan kemudahan pada sistem untuk melakukan orkestrasi dan pembuatan lingkungan *honeypot* serta mengamankan sistem *honeypot* dari penyerang yang berusaha mengambil alih sistem *honeypot* yang dimiliki oleh pengguna. Setiap serangan pada pengguna HaaS akan diarahkan menuju sistem HaaS milik penyedia melalui *proxy*/gateway pengguna. Semua hal yang dilakukan oleh penyerang pada sistem HaaS akan terekam pada sebuah log. Penyajian log dihimpun dalam satu halaman yaitu, halaman pengguna pada program aplikasi layanan HaaS. HaaS diharapkan dapat menyelesaikan permasalahan terkait pengelolaan *honeypot* yang apabila tidak dikelola dengan baik maka dapat merugikan pengguna dalam bentuk takeover sistem *honeypot* oleh penyerang dan sistem *honeypot* yang dihindari oleh penyerang.

II. LANDASAN TEORI

A. *Honeypot*

Layanan yang dibuat menyerupai sebuah sistem beserta layanan dan fitur yang terdapat pada jaringan bertujuan untuk menjebak dan menyulitkan penyerang. *honeypot* dapat diartikan sebagai sebuah misdirection terhadap penyerang yang berasumsi bahwa penyerang telah berhasil masuk ke dalam sistem untuk mengambil data pada jaringan. Peristiwa yang berlangsung pada *honeypot* sangat berguna untuk administrator jaringan sebagai salah satu bentuk notifikasi guna melakukan tindakan pencegahan sebelum terjadinya serangan yang sesungguhnya pada server yang sesungguhnya. *Honeypot* diklasifikasikan berdasarkan tingkat interaksinya [4]–[6], berikut klasifikasi *honeypot* berdasarkan tingkatan interaksi:

a. Interaksi rendah—*honeypot* mudah untuk diimplementasikan, dan dikonfigurasikan, dikarenakan desain *honeypot* yang sederhana dan tidak banyak fungsionalitas yang digunakan.
b. Interaksi menengah—interaksi pada klasifikasi ini lebih banyak namun masih belum menggunakan desain sistem operasi yang sesungguhnya. Desain *honeypot* menjadi lebih rumit, dimana *honeypot* memiliki fungsionalitas untuk merespon permintaan penyerang.
c. Interaksi tinggi—interaksi ini menerapkan desain sistem operasi yang sesungguhnya dengan seluruh fungsionalitas sistem yang tersedia untuk penyerang.

Terdapat dua keuntungan dalam menggunakan *honeypot*. Pertama, *honeypot* dapat memberikan bentuk pertahanan ke sistem yang sesungguhnya dengan cara menggunakan sistem palsu, dimana penyerang dapat dialihkan dari sistem yang sesungguhnya ke sistem *honeypot*. Kedua, informasi yang didapat dari serangan dapat digunakan untuk membangun mekanisme pertahanan yang lebih baik untuk kedepannya.

B. *Software-as-a-Service*

Aplikasi lengkap yang diberikan sebagai layanan kepada pengguna layanan. Pengguna layanan hanya perlu mengkonfigurasi beberapa parameter pada aplikasi. Penyedia layanan menangani semua infrastruktur, semua logika aplikasi, semua penerapan, dan segala sesuatu yang berkaitan dengan pengiriman produk atau layanan [3]. Perencanaan bisnis pada model SaaS terdiri dari tiga konsep utama yaitu:

a. SaaS merupakan cloud *service* model yang terdiri dari infrastruktur dan platform yang dibangun, dimiliki, dikelola, dan menjadi tanggung jawab penyedia layanan (aplikasi).
b. Model bisnis SaaS menggunakan *software* dengan metode *subscription*.
c. Model SaaS termasuk ke dalam konsep one-to-many yang menandakan bahwa terdapat paket standar layanan yang disediakan untuk sebanyak mungkin pengguna dan memberikan kustomisasi kepada setiap pengguna.

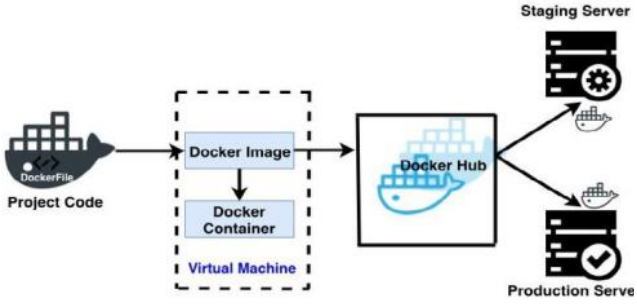
Menjalankan sebuah infrastruktur dan jaringan yang mendukung sebuah aplikasi tradisional dapat menjadi tugas yang sangat rumit bagi usaha kecil dan menengah [7]. Selain itu diperlukan biaya untuk membuat dan mengelola infrastruktur pada pengembangan aplikasi tradisional. Dengan menggunakan SaaS beban pengembangan dan pengelolaan layanan merupakan tanggung jawab penyedia. Aplikasi tradisional didesain untuk dapat dijalankan pada perangkat keras tertentu dan terkadang memiliki kebutuhan *software* sendiri seperti *operating system*, *driver*, dan yang lainnya. Kebutuhan tersebut membuat pengembangan aplikasi tradisional menjadi rumit ketika muncul kebutuhan aplikasi yang diminta untuk dapat berjalan pada berbagai macam perangkat keras dan lunak [1]. SaaS dijalankan pada lingkungan perangkat keras dan lunak pada sebuah server yang menjadikan proses pengembangan aplikasi menjadi lebih mudah. Pengguna mengakses SaaS melalui browser sehingga meminimalkan perintah yang harus dilakukan pengguna untuk melakukan pengembangan aplikasi. Dibandingkan dengan memperbaharui aplikasi pada setiap perangkat, SaaS hanya perlu memperbaharui aplikasi pada server.

Beberapa SaaS dapat digunakan pengguna secara mandiri tanpa perlu membutuhkan pelatihan dari penyedia SaaS. Informasi ini dapat diperoleh melalui dokumentasi yang tersedia dalam aplikasi. Penyedia SaaS pada umumnya hanya perlu menyediakan after sales support [3]. Keuntungan ini memungkinkan penyedia SaaS dapat menjangkau lebih banyak pengguna dengan mengalokasikan lebih banyak sumbe daya pada pemasaran daripada tim pelatihan pengguna.

C. Container

Virtualisasi berbasis kontainer adalah teknologi kernel-level yang mampu menjalankan banyak proses, masing-masing dalam lingkungan terisolasi. Kontainer memberikan banyak fleksibilitas dalam hal penerapan aplikasi [8]. Menggunakan kontainer tidak hanya aplikasi yang disebarkan, tetapi seluruh *stack* perangkat lunak. *Stack* perangkat lunak terbuat dari aplikasi itu sendiri, dependensi, sistem operasi, dan alat serta proses yang berjalan pada sistem operasi. Terdapat trend positif terkait teknologi kontainer diantaranya adalah docker dan kubernetes *cluster*.

Developer menggunakan Docker untuk mengemas aplikasi, kerangka kerja, dan dependensi aplikasi ke dalam sebuah image, dan kemudian mereka mengirimkan image tersebut kepada tester atau administrator [9]. Semua kontainer, apapun aplikasi yang berjalan di dalamnya, dapat diperlakukan dengan sama. Orkestrasi kontainerisasi docker dijelaskan pada Gambar 1.



Gambar. 1 Kontainerisasi Docker

DockerFile adalah file instruksi yang digunakan untuk membuat *container*. Docker Image adalah kumpulan lapisan isi dari kontainer, masing-masing dengan file JSON yang berisi metadata untuk lapisan tersebut [9]. Docker Image menggabungkan keseluruhan kebutuh kontainer untuk membuat aplikasi yang berjalan dimana image tersebut berinteraksi dengan kontainer ketika kontainer dijalankan.

Kubernetes adalah platform untuk mengelola aplikasi kontainer atau bisa disebut juga sebagai *container* orchestration [8]. kubernetes juga banyak memiliki fitur terhadap pengelolaan aplikasi kontainer termasuk autoscaling, penyebaran bergulir menghitung sumber daya, manajemen volume, dan lain-lain sama seperti desain yang ada pada docker bisa didesain dan dipakai di berbagai macam sistem, kubernetes juga mampu berjalan pada sistem baremetal, data center pada public cloud atau bahkan hybrid cloud.

Kubernetes terbagi menjadi master *node* dan worker *node* yang memiliki fungsi yang berbeda. Master *node* pada Kubernetes berfungsi sebagai master yang mengontrol keseluruhan unit dan *cluster*, mengatur workload serta komunikasi antar sistem [9]. Sedangkan worker *node* atau yang dikenal sebagai pekerja atau minion, merupakan mesin tempat *container* digunakan.

D. ELK Stack

Solusi analisis log yang lengkap, dibangun di atas kombinasi dari tiga aplikasi open source yaitu: Elasticsearch, Logstash, dan Kibana [10]. ELK menggunakan gabungan aplikasi open-source Elasticsearch untuk pencarian mendalam dan analisis data; Logstash untuk manajemen logging terpusat, yang mencakup pengiriman dan penerusan log dari beberapa server, elaborasi log, dan penguraian log; dan terakhir, Kibana untuk visualisasi data yang indah dan informatif [11].

Elasticsearch adalah mesin pencari dan analisis yang memungkinkan pencarian cepat dan terukur dalam lingkungan terdistribusi [11]. Elasticsearch bekerja dengan cara mengolah data, dimana pengguna dapat menyimpan dan melakukan pencarian berdasarkan data tersebut [10]. Elasticsearch menyembunyikan kompleksitas pencarian dibalik Apache Lucene dengan menyediakan REST API yang dibangun diatasnya, dimana REST API membuat kueri data yang diindeks menjadi lebih mudah, dan membuatnya tersedia untuk bahasa pemrograman apapun. Penggunaan REST API memperluas kemampuan Lucene dengan menyediakan analisis data secara real-time yang dibangun di atas data terstruktur dan tidak terstruktur berukuran besar yang didistribusikan di banyak server.

Logstash adalah aplikasi yang membantu mengumpulkan, mengurai, dan menganalisis berbagai macam data dan peristiwa terstruktur dan tidak terstruktur yang dihasilkan di berbagai sistem [11]. Logstash menyediakan plugin untuk terhubung ke berbagai jenis sumber data atau platform, yang dirancang untuk memproses log, peristiwa, dan sumber data tidak terstruktur secara efisien untuk didistribusikan ke berbagai tujuan. Logstash dapat mengambil input data dari protokol TCP/UDP, file, dan sistem manajemen log, seperti syslog-ng, rsyslog, dan berbagai macam jenis data lainnya [10], dimana aplikasi ini digunakan oleh administrator server untuk menganalisis peristiwa pada server.

Kibana adalah platform visualisasi data berlisensi Apache 2.0 open source yang membantu dalam memvisualisasikan segala jenis data terstruktur dan tidak terstruktur yang disimpan dalam indeks Elasticsearch [11]. Kibana dibuat menggunakan bahasa HTML dan JavaScript. Kibana menggunakan kemampuan pencarian dan pengindeksan dari Elasticsearch yang diekspos melalui REST API untuk menampilkan grafik bagi pengguna. Dari analisis bisnis hingga *real-time debugging*, Kibana menampilkan data yang didapat dengan histogram yang indah, peta geografis, diagram lingkaran, grafik, tabel, dan sebagainya. Kibana memudahkan untuk memahami volume data yang besar dengan antarmuka berbasis browser yang sederhana, memungkinkan administrator atau pengguna untuk membuat atau berbagi dashboard yang menampilkan perubahan informasi secara real-time.

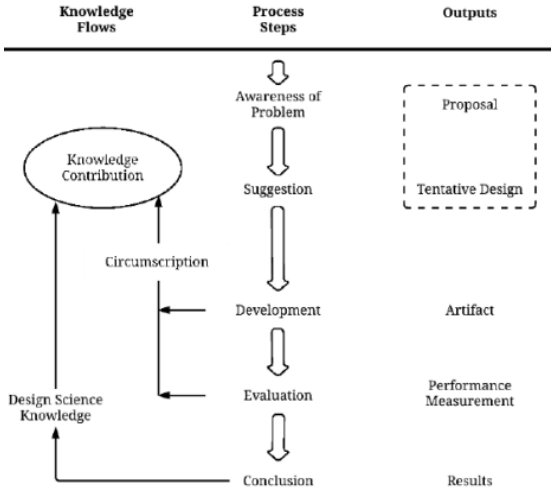
E. Penelitian Terkait

Dalam tesis Ahmed Elazazy [12] telah merumuskan aplikasi atau implementasi *honeypot* ke dalam wadah

docker bernama honey*proxy* sekaligus menyediakan kerangka kerja yang ditujukan untuk layanan berbasis cloud, sejalan dengan tesis Magnus Jonsson et al [13] juga mengimplementasikan *honeypot* dalam wadah terutama menggunakan high interaction *honeypot* yang menghasilkan prototipe *honeypot* modular sehingga dapat dikustomisasi lebih adaptif. Penelitian oleh Jorge Buzzio [14] juga menerapkan *honeypot* interaksi tinggi dalam sistem buruh pelabuhan dan diuji di lingkungan nyata dan menunjukkan keefektifannya setelah analisis di tingkat jaringan dan tingkat host dan menggunakan alat seperti VirusTotal. Sebuah studi kasus juga telah diterapkan oleh Rasmi-Vlad Mahmoud di sebuah universitas menggunakan Docker dan menunjukkan bahwa *honeypot* dengan interaksi rendah dapat meminimalkan risiko. penelitian Jafar Haadi [7] melakukan implementasi sistem *honeypot* Proof-of-Concept sebagai layanan. Validasi dan analisis efektivitas dan keamanan telah dilakukan oleh server Dubravo [15] dalam menjawab tantangan *honeypot* yang gesit dan fleksibel. Menggabungkan keunggulan tersebut, penelitian ini mencoba menerapkan konsep *service* atau sering disebut dengan *software* as a *service* , khususnya untuk *honeypot*s.

III. METODOLOGI PENELITIAN

Desain penelitian didasarkan pada pengetahuan yang disimpulkan dalam bentuk pengembangan, pemodelan, pengembangan teori, teknik dan metode pemetaan artefak yang telah diproduksi atau pemenuhan produk terhadap persyaratan fungsional yang telah ditentukan [16], [17]. Penelitian ini menggunakan metodologi Design Science Research Methodology (DSRM). Siklus DSRM sebagaimana direpresentasikan pada Gambar 2 merupakan metode yang menentukan dan melakukan penelitian dengan tujuan akhir berupa rekomendasi artefak. Metode ini bertujuan untuk memecahkan masalah tertentu agar mendapatkan solusi yang efektif, meskipun solusi yang diperoleh tidak optimal.



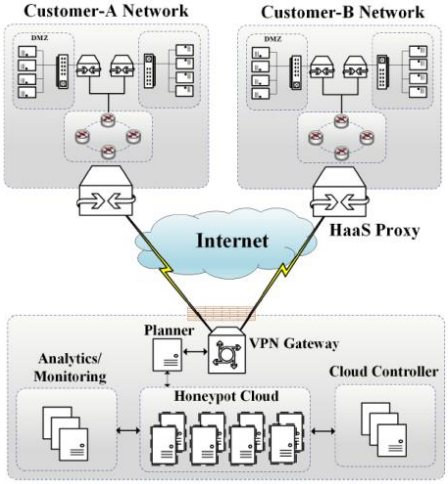
Gambar. 2 DSRM *process model cycle*

A. *Awareness of Problem*

Perumusan masalah digunakan untuk merumuskan masalah dengan eksperimentasi dan eksplorasi sebagai pola pengembangan penelitian. Identifikasi masalah dimulai dengan mengeksplorasi topik *honeypot*s dan pengembangannya ke depan. Studi literatur dilakukan mengenai kemampuan *honeypot* dengan kelebihan dan kekurangan *honeypot*. Proses identifikasi masalah diperkuat dari hasil penelitian yang telah dilakukan sebelumnya. Hasil dari proses identifikasi masalah adalah bahwa *honeypot* membutuhkan pengelolaan dan pemeliharaan yang intensif agar dapat berfungsi dengan baik. Jika *honeypot* tidak dipelihara, penyerang dapat mengambil alih *honeypot*, menyebabkan kerugian bagi pengguna *honeypot*.

B. *Suggestion*

Berisi saran atau solusi yang ditawarkan untuk mengatasi masalah yang telah ditetapkan pada tahap kesadaran masalah. Pola yang digunakan pada tahap saran adalah memodelkan solusi yang ada. Solusi yang diberikan merupakan hasil adaptasi dari solusi pada penelitian sebelumnya untuk memecahkan masalah pada tahap awareness of problem. Solusi yang dimaksud adalah gabungan dari beberapa penelitian yang ada [1], [7], [18]. Solusi yang dimaksud adalah membuat model penggunaan *honeypot* berbasis SaaS untuk memudahkan pengguna memilih dan mengatur *honeypot* untuk digunakan dalam jaringan. Hasil tahap saran berupa desain atau model sistem HaaS yaitu *honeypot* yang menggunakan model layanan SaaS yang dibangun menggunakan Kubernetes *Cluster*.



Gambar. 3 Model *Honeypot*-as-a-*Service*

Gambar 3, *Honeypot* tersimpan didalam sebuah sistem Cloud yang memiliki fungsi analitik dan monitoring, kemudian terdapat kontroler dan planner pada sistem cloud tersebut. Akses terhadap sistem Cloud disalurkan melalui VPN Tunnel yang terhubung pada VPN Gateway pada Cloud. Pengguna terhubung dengan Cloud *Honeypot* melalui *Proxy* yang bertujuan untuk melakukan redireksi seluruh jenis traffic kepada *Honeypot* Cloud.

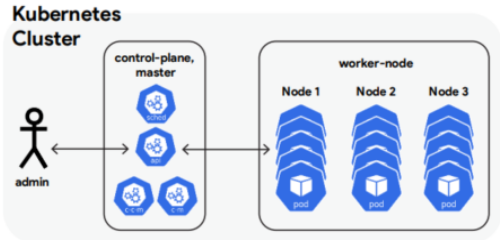
Sumber daya yang digunakan *node* berasal dari perangkat server yang berada pada Datacenter

Laboratorium Keamanan Siber Poltek SSN. Jaringan untuk *cluster* menggunakan network interface yang terdapat pada datacenter terhubung dengan jaringan Poltek SSN. Tabel 1 menjelaskan mengenai pembagian jaringan dan label pada *node* yang terdapat pada *cluster*. Pembagian label bertujuan untuk membagi fungsi kerja masing-masing *node*. *Control-plane*/master merupakan roles atau label yang diberikan kepada sebuah *node* untuk melakukan penjadwalan, pemantauan semua hal yang terjadi pada *cluster*. Label worker merupakan tag yang diberikan kepada *node* yang bergabung ke dalam sebuah kluster yang akan bekerja untuk menjalankan setiap perintah dari *control-plane*.

TABEL I
NETWORK RESOURCES

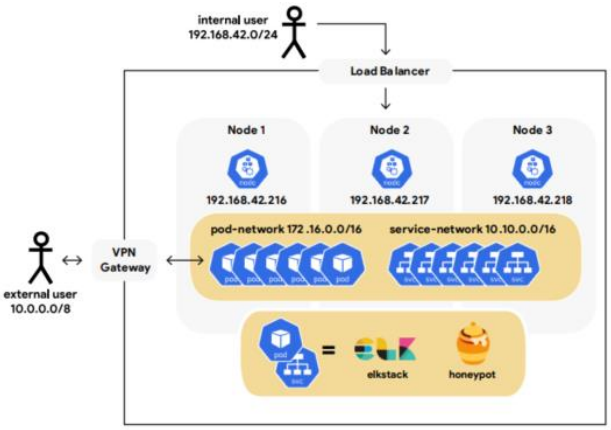
No	Host	IP address	Tag
1	Kmaster	192.168.42.216	Master, controlplane
2	Kworker1	192.168.42.217	Worker
3	Kworker2	192.168.42.218	Worker

1) Perancangan Arsitektur Kubernetes: Gambar 4 merupakan abstraksi secara keseluruhan mengenai komponen utama pada Kubernetes *Cluster*. Pembuatan dan perubahan sebuah konfigurasi pada *control-plane* maupun *worker* menggunakan api. Dalam Kubernetes *Cluster*, pembagian *pod* pada setiap *node* berlangsung secara abstrak dimana setiap *pod* dapat berhubungan dengan satu sama lain tanpa terhalang batasan jaringan pada *node* [19]. Pada Gambar 5 alamat ip dari *pod* dan *service* terdapat pada satu jaringan internal yang sama. Hal ini dapat menyebabkan permasalahan apabila terdapat sebuah *honeypot* yang berhasil diambil alih oleh penyerang besar kemungkinan penyerang tersebut dapat mengambil alih *pod* atau *service* lainnya sehingga perlu adanya segmentasi pada *pod* dan *service* dalam Kubernetes *Cluster* untuk mengurangi kerusakan pada kluster apabila terdapat serangan.

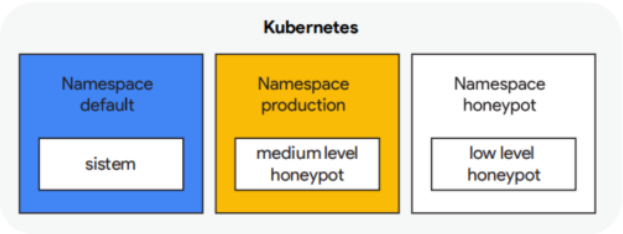


Gambar. 4 Model *Honeypot*-as-a-*Service*

Dalam Kubernetes *Cluster* terdapat fungsi namespace yang menyediakan mekanisme untuk mengisolasi grup sumber daya dalam satu *cluster* [19]. *Pod* dan *service* terrmasuk ke dalam grup sumber daya tersebut dimana, *pod* dan *service* ini dapat diduplikasikan antar namespace tanpa adanya konflik. Ruang lingkup penggunaan namespace hanya berlaku untuk objek individual seperti *Deployment*, *Service* s, dll dan bukan untuk objek yang bersifat terbagi pada *cluster* seperti StorageClass, *Nodes*, PersistentVolumes, dan lain-lain [19]. *Cluster* nantinya akan disegmentasikan ke dalam tiga namespace yaitu default, production, dan *honeypot*. default merupakan tempat dimana sistem pendukung HaaS berjalan, production adalah tempat layanan untuk medium level interaction *honeypot* yang dispesifikasikan oleh pengguna, dan *honeypot* adalah tempat dimana layanan *low level interaction honeypot* dikelompokkan. Penggunaan namespace ini bertujuan untuk mengisolasi beberapa bagian pada Kubernetes *Cluster* sesperti ditunjukkan dalam Gambar 6. Pada namespace default sistem akan menjalankan beberapa layanan diantaranya yaitu SQL server, monitoring dengan ELK, dan dashboard kubernetes. Production untuk menampung docker image milik pengguna. *Honeypot* untuk menampung *honeypot* yang akan digunakan oleh pengguna.

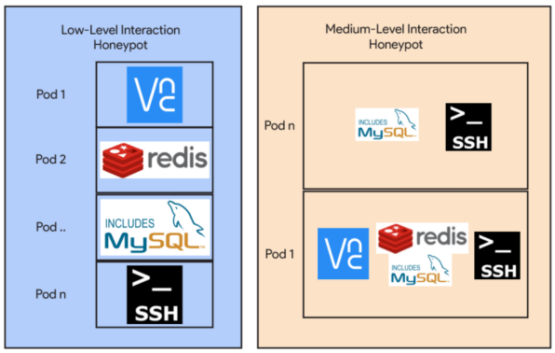


Gambar. 5 Rancangan HaaS pada Kubernetes



Gambar. 6 Pembagian Namespaces pada *cluster*

2) Perancangan Arsitektur Kubernetes: Berdasarkan perancangan sistem, layanan *honeypot* terbagi menjadi dua skema, *low level interaction honeypot* dan *medium level interaction honeypot*. Kedua skema tersebut dikategorikan berdasarkan jumlah layanan yang digunakan oleh pengguna. *Low level interaction honeypot* pada HaaS merupakan sebuah *honeypot* yang hanya akan mensimulasikan satu jenis layanan dan untuk *medium level interaction honeypot* merupakan layanan *honeypot* yang memiliki dua atau lebih jenis layanan yang akan disimulasikan. Gambar 7 memvisualisasikan perbedaan antara *low level interaction honeypot* dan *medium level interaction honeypot* pada HaaS. Jenis layanan yang akan disimulasikan yaitu: ssh, telnet, rdp, vnc, mysql, redis, dan web *service* menggunakan Apache. Pemilihan layanan tersebut bertujuan untuk mempermudah implementasi layanan tersebut ke dalam bentuk konfigurasi kubernetes.



Gambar. 7 visualisasi low dan medium level *honeypot*

C. *Development*

Proses yang dilakukan pada tahap *development* adalah sintesis antara pengetahuan dari studi literatur dengan permasalahan yang telah didefinisikan ke dalam bentuk artefak sebagai solusi dari permasalahan. Pada tahap ini, dilakukan implementasi dan pengembangan desain yang telah dibuat pada tahap *suggestion*. Tahap *development* dimulai dengan membuat daftar fitur dan layanan yang perlu disimulasikan oleh *honeypot* seperti terlampir dalam Tabel 2 yang merupakan kebutuhan fungsional dan Tabel 3 yang menunjukkan kebutuhan non fungsional.

TABEL II
KEBUTUHAN FUNGSIONAL SISTEM

No	Fungsional Sistem
1	Sistem *Honeypot*-as-a-*Service* dapat melakukan orkestrasi untuk pengelolaan dan penjadwalan pada kontainer
2	Sistem *Honeypot*-as-a-*Service* memiliki *honeypot* yang dapat menerima dan mencatat serangan
3	Sistem *Honeypot*-as-a-*Service* memiliki fungsi logging
4	Sistem *Honeypot*-as-a-*Service* memiliki dashboard untuk manajemen *honeypot*

TABEL III
KEBUTUHAN NON FUNGSIONAL SISTEM

No	Kebutuhan Non-Functional
1	Sistem *Honeypot*-as-a-*Service* dibangun dengan menggunakan Kubernetes untuk orkestrasi kontainer.
2	Sistem *Honeypot*-as-a-*Service* menggunakan *honeypot* yang bersifat open-source
3	Sistem *Honeypot*-as-a-*Service* menggunakan datacentre sebagai lingkungan implementasi
4	Sistem *Honeypot*-as-a-*Service* memiliki mekanisme untuk menjaga ketersediaan, skalabilitas, dan performa.

Pada HaaS terdapat beberapa infrastruktur yang dikembangkan diantaranya: penggabungan opensource *Honeypot*, autentikasi *honeypot* dan pengguna, Analisis data *honeypot* pada server menggunakan ELK Stack, orkestrasi dengan Kubernetes *Cluster*, *proxy* atau *gateway honeypot*, dan halaman *dashboard* pengguna selain itu juga mensimulasikan *honeypot* berdasar pada laporan tren serangan siber yang dihimpun oleh ENISA, OWASP, dan Sophos [19]–[21]. *Honeypot* yang telah dikembangkan kemudian akan diuji coba untuk memastikan *honeypot* dapat berjalan dengan semestinya. Kemudian dikemas dalam sebuah image atau kontainer untuk dapat dijalankan pada *cluster*. Setiap *node honeypot* nantinya akan menghasilkan data atau log yang kemudian akan dianalisis menggunakan ELK Stack untuk ditampilkan pada halaman atau dashboard pengguna.

1) Instalasi dan Konfigurasi Kubernetes Cluster: Deployment Kubernetes Cluster pada setiap *node* dilakukan secara otomatis melalui *shell* skrip untuk mempermudah penambahan *node* baru pada *cluster*, apabila terjadinya ekspansi pada sistem. Pemasangan kubernetes pada *node* menggunakan *toolbox* kubernetes.io. Konfigurasi topologi jaringan pada *cluster* terdapat pada parameter inisialisasi *cluster* menggunakan kubeadm. Topologi pada kubernetes diatur menggunakan *Container* Network Interfaces (CNI) [21] yang bertanggung jawab untuk memberikan alamat IP kepada *pod* dan *service* yang berjalan di dalam Kubernetes dan menjadwalkan *worker node* untuk merutekan paket sesuai dengan model topologi jaringan.

```
$ sudo kubeadm init --pod-network-cidr=172.16.0.0/16 --service-cidr=10.10.0.0/16 --cri-socket /run/cri-dockerd.sock
```

Gambar. 8 kode sumber perintah inisialisasi kuberrrnates

Pemilihan Alamat IP *pod* pada Kode Sumber Gambar 8 diambil dari kumpulan IP yang dibuat ketika dilakukan instalasi dan dipilih pada rentang jaringan sesuai dengan standar RFC1918. Alamat IP tersebut dikenal dengan *pod* network CIDR(Classless InterDomain Routing) pada kubernetes. Konfigurasi parameter jaringan tersebut adalah dengan menggunakan argumen --*pod*-network-cidr untuk *pod* dan –*service* -networkcidr untuk layanan menggunakan format notasi CIDR.

Jaringan yang digunakan oleh *pod* dan *service container* terdapat pada abstraksi yang tersebar ke seluruh *cluster* dan mengatur bagaimana lalu lintas jaringan terjadi di seluruh *node* dalam *cluster*. Secara garis besar, semua *pod* dan *service* dalam *cluster* berada pada subnet yang sama. Pada saat inisialisasi kubernetes dengan menggunakan kubeadm, antarmuka jaringan logis disediakan pasangan veth pada master *node*. Antarmuka veth ini digunakan oleh worker *node* sebagai metode berhubungan dengan *node* lainnya dan juga dengan master *node* Gambar 9 menunjukkan *cluster* sudah memiliki *container* network interface.

```
5: veth3bd25b0@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 5e:55:41:0e:3c:ae brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::5c55:41ff:fe0e:3cae/64 scope link
       valid_lft forever preferred_lft forever
7: veth721c9e8@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 6e:92:d4:97:b7:20 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::6c92:d4ff:fe97:b720/64 scope link
       valid_lft forever preferred_lft forever
```

Gambar. 9 *Container* Network Interface

Dalam menjaga kesehatan *container* serta keberlangsungan layanan pada *cluster*, perlu adanya fitur *recovery* apabila terjadi kegagalan. HaaS menggunakan dua mekanisme diantaranya: mekanisme *failover* dan mekanisme *autoscaling*. Mekanisme failover menyediakan fitur *Deployment* untuk mengatasi *masalah failure*. Fitur ini memastikan bahwa layanan yang berjalan akan selalu berjalan dengan semestinya secara periodik. Pada mekanisme *autoscaling* parameter yang digunakan agar Kubernetes melakukan *scale up* atau *Scale down* adalah

CPU, dimana masing-masing *pod* akan diberikan sumber daya CPU sejumlah tertentu, lalu diberi batasan dalam persen sehingga, ketika CPU *load* melebihi parameter tertentu, maka Kubernetes akan melakukan *scaling up* atau *Scaling down* secara otomatis [19]. Pada proses scaling, Kubernetes akan memeriksa keadaan masing-masing *node* apakah *node* yang berada di dalam *cluster* mampu menjalankan layanan sesuai permintaan jika tersedia *node* yang memiliki sumber daya yang sesuai maka *node* tersebut akan dipilih untuk menjalankan layanan.

2) Konfigurasi Layanan pada Cluster

Terdapat beberapa parameter yang harus disesuaikan pada pembuatan konfigurasi layanan yaitu penggunaan API stabil dan *up to date* dan melakukan penyimpanan konfigurasi ke dalam version control sebelum dilakukan push ke dalam *cluster*. Hal ini bertujuan untuk mengembalikan perubahan konfigurasi jika diperlukan dan yang terakhir adalah keseluruhan konfigurasi ditulis menggunakan format UAML [19]. Untuk konfigurasi medium level interaction *honeypot* difokuskan untuk memasang *honeypot* pada namespace production seperti terlampir pada Gambar 10 dan mendefinisikan kebutuhan layanan ssh, telnet, rdp, vnc, mysql, redis, postgres, mongodb, http, dan api server seperti dalam cuplikan kode Gambar 11 Konfigurasi tersebut menggunakan fitur *LoadBalancer* untuk membuka akses layanan kepada pengguna maupun administrator jaringan *cluster*.

```
spec:
  containers:
    - name: server
      image: sierrasoftworks/honeypot:latest
      imagePullPolicy: IfNotPresent
      resources:
        requests:
          cpu: 50m
          memory: 100Mi
        limits:
          cpu: 500m
          memory: 500Mi
      env:
        - name: ENVIRONMENT
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
      ports:
        - name: ssh
          containerPort: 2222
          protocol: TCP
        - name: telnet
          containerPort: 2323
          protocol: TCP
        - name: rdp
          containerPort: 3389
          protocol: TCP
        - name: vnc
          containerPort: 5900
          protocol: TCP
        - name: mysql
          containerPort: 3306
          protocol: TCP
        - name: redis
          containerPort: 6379
          protocol: TCP
        - name: postgres
          containerPort: 6379
          protocol: TCP
        - name: mongodb
          containerPort: 27017
          protocol: TCP
        - name: http
          containerPort: 8081
          protocol: TCP
        - name: api
```

Gambar. 10 Konfigurasi *Deployment Honeypot*.

```
apiVersion: v1
kind: Service
metadata:
  name: {user}-honeypot-server
spec:
  selector:
    app.kubernetes.io/name: {user}-honeypot
    app.kubernetes.io/instance: {user}-honeypot-server
  ports:
    - name: http
      port: {port}
      targetPort: api
      protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: honeypot-ssh
spec:
  selector:
    app.kubernetes.io/name: {user}-honeypot
    app.kubernetes.io/instance: {user}-honeypot-server
  type: LoadBalancer
  externalTrafficPolicy: Local
  ports:
    - name: ssh
      port: 22
      targetPort: ssh
      protocol: TCP
    - name: telnet
```

Gambar. 11 Konfigurasi layanan *Honeypot*.

Konfigurasi layanan untuk model *low level interaction honeypot* mempunyai perbedaan konfigurasi terletak pada *deployment low level interaction honeypot* dimana setiap layanan memiliki *pod* nya masing-masing neskipun sama-sama menggunakan konfigurasi layanan dengan *LoadBalancer*. Untuk mempermudah pembuatan konfigurasi, Gambar 12 menunjukkan otomatisasi pembuatan konfigurasi layanan dengan menggunakan *dictionary* yang berisikan nama layanan, *image* layanan, jumlah replika, dan *port* yang digunakan.

```
1.  def writeConfig(**kwargs):
2.      template = """
3.      apiVersion: v1
4.      kind: pod
5.      metadata:
6.        name: {name}
7.      spec:
8.        replicas: {replicas}
9.        template:
10.         metadata:
11.           labels:
12.             run: {name}
13.         spec:
14.           containers:
15.           - name: {name}
16.             image: {image}
17.             ports:
18.             - containerPort: {port}
19.
20.     with open('config.yaml', 'w') as yfile:
21.         yfile.write(template.format(**kwargs))
22.
23. # usage:
24. writeConfig(name="[Name]", image="[image]", replicas="[count], port="[port]"")
```

Gambar. 12 *Service configuration python script*

3) *Konfigurasi Monitoring dengan ELK:* Pada layanan *monitoring* terdapat dua komponen utama dan dua komponen *agent* untuk *monitoring cluster* yaitu *elasticsearch* dan *logstash*, serta *filebeat* dan *metricbeat*. *Elasticsearch* merupakan tempat data atau log disimpan, *logstash* berguna untuk mengagregasi data atau log ke dalam format yang dapat disimpan pada *elasticsearch*. semantara itu, *Metricbeat* berguna untuk mengumpulkan *metric* status dari layanan maupun node seperti kondisi CPU dan RAM kemudian, *filebeat* merupakan *agent* yang bertugas untuk mengumpulkan *node* mengirimkannya kepada *logstash*. Gambar 13 merupakan konfigurasi dari *elasticsearch*. *Logstash* bertindak sebagai *agregator* yang menerima log dalam bentuk dasar, dan memformatnya ke

dalam format yang telah didefinisikan sebelumnya oleh *Elasticsearch* ditunjukkan dalam Gambar 14.

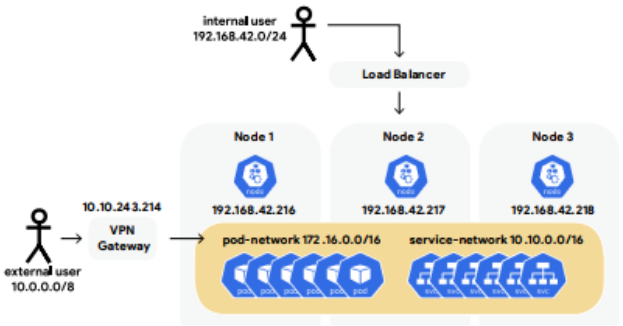
```
clusterName: "elasticsearch"
nodeGroup: "master"

masterService: ""

roles:
  master: "true"
  ingest: "true"
  data: "true"
  remote_cluster_client: "true"
  ml: "true"

replicas: 3
minimumMasterNodes: 2

image: "docker.elastic.co/elasticsearch/elasticsearch"
imageTag: "7.17.3"
imagePullPolicy: "IfNotPresent"

labels: {}
resources:
  requests:
    cpu: "1000m"
    memory: "2Gi"
  limits:
    cpu: "1000m"
    memory: "2Gi"

networkHost: "0.0.0.0"

volumeClaimTemplate:
  accessModes: ["ReadWriteOnce"]
  resources:
    requests:
      storage: 30Gi
```

Gambar. 13 Konfigurasi Elasticsearch.

```
replicas: 1
logstashPatternDir: "/usr/share/logstash/patterns/"
image: "docker.elastic.co/logstash/logstash"
imageTag: "7.17.3"
imagePullPolicy: "IfNotPresent"
imagePullSecrets: []

podAnnotations: {}

# additionals labels
labels: {}

logstashJavaOpts: "-Xmx1g -Xms1g"

resources:
  requests:
    cpu: "100m"
    memory: "1536Mi"
  limits:
    cpu: "1000m"
    memory: "1536Mi"

volumeClaimTemplate:
  accessModes: ["ReadWriteOnce"]
  resources:
    requests:
      storage: 1Gi

rbac:
  create: false
  serviceAccountAnnotations: {}
  serviceAccountName: ""
  annotations:

podSecurityPolicy:
  create: false
  name: ""
  spec:
    privileged: false
    fsGroup:
      rule: RunAsAny
    runAsUser:
      rule: RunAsAny
    seLinux:
      rule: RunAsAny
    supplementalGroups:
      rule: RunAsAny
    volumes:
      - secret
      - configMap
      - persistentVolumeClaim

persistence:
  enabled: false
  annotations: {}
```

Gambar. 14 Konfigurasi logstash.

4) *Menghubungkan Cluster melalui VPN:* Pembuatan VPN *Gateway* dilakukan untuk mengisolasi *traffic* dari jaringan pengguna dan untuk memberikan destinasi alamat kepada pengguna. VPN *Gateway* dapat difungsikan sebagai mode akses kontrol kepada pengguna, dan juga untuk melakukan *monitoring* akses kepada sistem. HaaS menggunakan *addon* jaringan yang mendukung *LoadBalancer*, sehingga setiap *pod* dapat memiliki *port* dan ip externalnya masing-masing. *Routing tables* yang terdapat pada VPN *Gateway* akan menghubungkan pengguna dengan alamat yang terdapat pada *cluster* yang telah dipublikasi melalui *LoadBalancer*, sehingga pengguna dapat mengakses jaringan *honeypot* melalui jaringan internal mereka sendiri. Diagram akses pada HaaS dapat dilihat pada Gambar 15.



Gambar. 15 Diagram akses pada HaaS.

HaaS memanfaatkan *add-on* jaringan *Flannel* dan *openvpn* sebagai VPN *Gateway* serta server VPN. Gambar 16 merupakan perintah biner kube*proxy* untuk menggunakan konfigurasi kube-config untuk menghubungkan VPN dengan *proxy* sekaligus konfigurasi penanganan paket menggunakan iptables dengan menambah aturan SNAT untuk meneruskan paket konfigurasi. Konfigurasi dilakukan untuk mengidentifikasi *node* etcd yang menyimpan sertifikat publik *cluster* dan memberikan parameter *runtime* yang tepat. parameter /*cluster*.local/network menentukan batasan jaringan dan parameter --public-ip adalah nilai dari alamat ip yang dimiliki oleh VPN *Gateway*. *Script* juga menghubungkan *node* VPN dengan jaringan *overlay Flannel*, namun hubungan tersebut perlu dilakukan konfigurasi tambahan terkait *routing*. Dikarenakan, alamat ip layanan adalah sebuah alamat virtual, dan dikelola oleh *kube-proxy* dengan manajemen rute melalui ip table. Perlu adanya penghubung antara VPN dengan *kube-proxy* untuk memberikan akses ke luar dan ke dalam *cluster*. Pada *kube-proxy* terdapat konfigurasi kubeconfig untuk menghubungkan *kube-proxy* kepada API sehingga VPN dapat terhubung dengan menggunakan API tersebut [19]. Penggunaan alamat ip 10.10.243.214 merupakan alamat ip dari VPN Gateway, alamat ip 10.0.0.0/8 merupakan paket yang bersumber dari luar dengan destinasi ke dalam ip *cluster* yaitu 172.16.0.0/16 untuk *pod* dan 10.10.0.0/16 untuk layanan.

```
$ sudo flannel --control-plane-endpoints 192.168.42.216:6443, -etcd-prefix
/cluster.local/network -etcd-cafile /opt/flannel-config/ca_cert.crt -etcd-certfile
/opt/flannel-config/cert.crt -etcd-keyfile /opt/flannel-config/key.pem -public-ip
10.10.243.214
$ kube-proxy --kubeconfig=./kube-config/config.yaml --bind-address=10.10.243.214 --
cluster-cidr=10.10.0.0/16 -proxy-mode=iptables --masquerade-all
$ push "route 10.10.0.0 255.255.0.0"
$ iptables -t nat -I POSTROUTING -s 10.0.0.0/8 -d 172.16.0.0/16,10.10.0.0/16 -j SNAT --
to-source 10.10.243.214
```

Gambar. 16 Perintah Menghubungkan Flannel dengan VPN.

5) *Pembuatan Dashboard HaaS:* Pembuatan *dashboard* pada HaaS bertujuan untuk memberikan kemudahan bagi pengguna dalam melakukan pendaftaran akun, pemilihan *honeypot*, dan memantau *honeypot* tersebut. Pembuatan dilakukan dengan menggunakan *framework* HTML, PHP, dan MySQL. Pembuatan *dashboard* dilakukan untuk memenuhi analisis kebutuhan fungsional dimana, pengguna dapat mendaftarkan akun, memilih *honeypot*, dan memantau *honeypot* tersebut. Berdasarkan kebutuhan tersebut, perlu dipetakan pemenuhan fungsi dan kebutuhan pada *dashboard* yang tertera pada Tabel 4.

TABEL IV
FILE UNTUK PEMENUHAN NON FUNGSIONAL SISTEM

Nama	Fungsi	Kebutuhan
signup.php	Pengguna melakukan registrasi akun	Registrasi akun
login.php	Pengguna masuk ke dalam dashboard menggunakan kredensial yang telah didaftarkan	Masuk ke dalam dashboard
logout.php	Pengguna keluar dari akun yang digunakan	Keluar dari dashboard
welcome.php	Halaman utama pengguna dimana, pengguna dapat memilih jenis *honeypot* yang akan digunakan.	Halaman utama pengguna, dan memilih *honeypot*
config.php	Sebagai penghubung backend sistem	
stats.php	Menampilkan log yang terdapat pada *honeypot*	Pemantauan *honeypot*

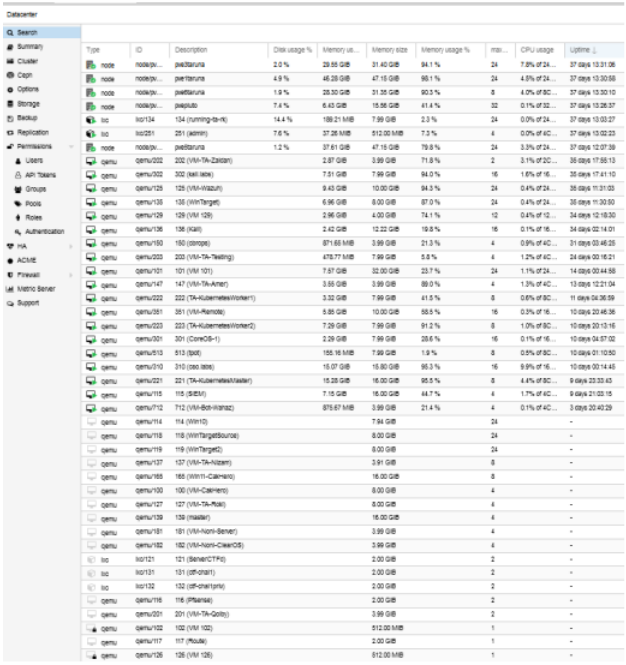
D. Evaluation

Proses yang dilakukan pada tahap evaluation adalah menentukan seberapa baik kinerja dari artefak berdasarkan metode empiris yang digunakan. Tahap ini merupakan kesempatan untuk melakukan perbaikan terhadap artefak berdasarkan pengalaman yang didapatkan selama tahap sebelumnya. Hal ini disebut juga circumscription pada F cycle.

Pada tahap evaluation juga dilakukan Pengujian dilakukan dengan lingkungan virtual Proxmox VE pada datacenter Jurusan Keamanan Siber Politeknik Siber dan Sandi Negara. Penggunaan lingkungan virtual pada Proxmox VE dinilai cukup untuk mensimulasikan jaringan pengguna dikarenakan, pada saat Gambar 17 ditangkap, terdapat 20 Virtual Machine dan 15 *Container* yang sedang berjalan.

Cloud tesing dilakukan Pengujian yang terdiri dari empat jenis yaitu Availability Testing, Performance Testing, Functional Testing dan Scenario testing. Availability Testing menggunakan white-box testing dengan pendekatan menggunakan pengetahuan tentang struktur data internal, aliran logika fisik, dan arsitektur pada tingkat kode sumber sistem [22]. Pengujian ini dilakukan dengan melihat pengujian dari sudut pandang pengembang menggunakan *checklist* kriteria yang telah diolah kembali [23]–[25].

Performance testing dilakukan menggunakan blackbox testing dengan melakukan Stress Test, Load Test, dan Scalability Test pada komponen sistem. Pengujian dilakukan dengan fokus utama pada perangkat yaitu CPU, Memory, Disk I/O, dan Penggunaan Jaringan [24], [25]. Pengujian load testing bertujuan untuk menguji kemampuan fungsional sistem ketika terdapat kebutuhan akses yang besar [24].



Gambar. 17 Lingkungan Proxmox VE.

Functional Testing dilakukan dengan menyebarkan kuesioner kepada responden. Kuesioner pengujian menggunakan skala Likert untuk memperoleh persentase kepuasan pengguna terhadap sistem HaaS. Skala Likert yang digunakan memiliki rentang penilaian 1 dan 5. Perhitungan pada skala likert dilakukan dengan menggunakan formula 1.

$$X = \frac{\sum(a)}{b} \times 100\% \quad (1)$$

TABEL V
PARAMETER FUNCTIONAL TESTING

No	Test parameters	expected results
1	Apakah layanan dapat beradaptasi dengan lingkungan tertentu?	ya
2	Apakah terdapat banyak konfigurasi yang diperlukan?	tidak
3	Apakah kustomisasi mungkin?	ya
4	Apakah (banyak) perangkat lunak khusus tambahan diperlukan?	tidak
5	Apakah platform yang diperlukan didukung?	ya
6	Apakah mungkin untuk menghubungkan/mengintegrasikan layanan dengan sistem lain?	ya
7	Apakah tersedia cukup manual dan/atau kursus?	ya

Nilai X merupakan skor atau persentase dari keseluruhan pertanyaan, dimana a merupakan nilai yang didapat dari

masing-masing pertanyaan dan b merupakan total kemungkinan nilai yang didapat apabila a bernilai maksimum (lima). Terdapat 7 pertanyaan yang diajukan kepada responden, pertanyaan tersebut bersumber dari [23] yang telah diolah kembali untuk menyesuaikan dengan lingkungan penelitian dan dapat dilihat pada Tabel 5. Pada parameter pengujian erdapat kolom "Hasil yang diharapkan", untuk kolom yang bernilai "Tidak", Kolom tersebut menandakan bahwa hasil nilai pertanyaan tersebut bersifat inverse dari nilai sesungguhnya dimana skala 1 merupakan nilai maksimum(5) yang dapat diperoleh dari pertanyaan tersebut.

Pengujian dengan skenario bertujuan untuk menguji keefektifan dari *honeypot* yang berada pada sistem HaaS berhasil dalam menjebak serangan. Pengujian dilakukan dengan menggunakan kuesioner yang disebar kepada para responden atau tester yang telah mencoba melakukan serangan kepada sistem HaaS. Terdapat 20 orang responden yaitu, Taruna Satria Politeknik Siber dan Sandi Negara yang memiliki sertifikasi kompetensi internasional. Tabel 6 mendefinisikan lingkungan pengujian dimana terdapat tiga alamat pada jaringan yang akan diserang oleh responden. Pengujian ini dilakukan secara Blind Test dimana responden tidak mendapatkan informasi target serangan.

TABEL VI
DETAIL TARGET SERANGAN

No	Alamat Mesin	Keterangan	OS
1	192.168.42.170	*Honeypot* Dionaea	Ubuntu 20.04
2	192.168.42.156	HaaS-*Proxy*	Ubuntu 20.04
3	192.168.42.191	Ubuntu Server	Ubuntu 20.04

Hasil yang diharapkan dari scenario test dengan pendekatan blind testing ini adalah 2/3 responden dapat menjawab dengan benar pertanyaan yang terdapat pada kuesioner yaitu: PC 1 merupakan sebuah *honeypot*, PC 2 bukan sebuah *honeypot*, dan PC 3 bukan sebuah *honeypot*.

TABEL VII
PARAMETER SCENARIO TESTING

No	Pertanyaan	Expected result
1	Apakah menurut anda PC 1 (.170) merupakan sebuah *Honeypot*?	>66%
2	Apakah menurut anda PC 2 (.156) merupakan sebuah *Honeypot*?	>66%
3	Apakah menurut anda PC 3 (.191) merupakan sebuah *Honeypot*?	>66%

Skenario pengujian didefinisikan pada Tabel 7 yang diolah kembali dari [26]. Ketiga *honeypot* tersebut memiliki fungsi yang berbeda diantaranya; PC 1 berfungsi sebagai *honeypot* tradisional yang sesungguhnya dengan tingkat interaksi rendah, *honeypot* tersebut sering digunakan dikarenakan performa *honeypot* tersebut; PC 2 merupakan layanan *honeypot* yang terhubung dengan sistem; PC 3 Merupakan variable kontrol pada penelitian.

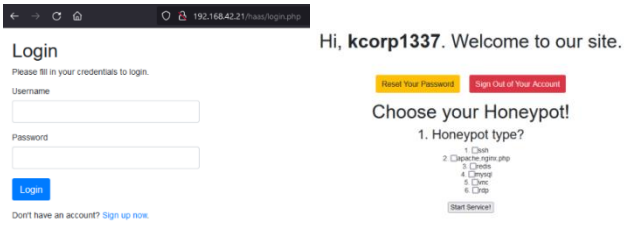
E. Conclusion

Tahap ini merupakan tahap terakhir dari siklus penelitian dalam DSRM. Hasil dari tahap ini merupakan penyampaian solusi yang dapat menjawab rumusan masalah dalam penelitian.

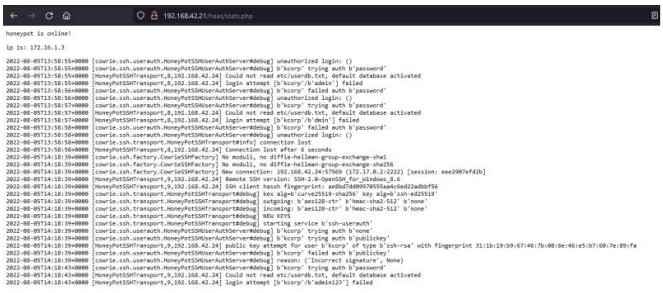
IV. HASIL DAN PEMBAHASAN

A. Dashboard HaaS

Gambar 18 merupakan visualisasi halaman yang digunakan pengguna untuk melakukan sistem. Gambar tersebut juga memberikan pilihan terhadap pengguna mengenai jenis *honeypot* yang akan digunakan. Kemudian statistik atau log dari *honeypot* akan ditampilkan pada halaman stats.php yang divisualisasikan pada Gambar 19 Data yang ditampilkan pada halam tersebut dihasilkan dari hasil parsing log yang dihasilkan oleh sistem HaaS.



Gambar. 18 Tampilan halaman HaaS.



Gambar. 19 log *honeypot*

B. HaaS Proxy

Pada model HaaS terdapat sebuah *proxy* yang berfungsi melakukan redireksi semua traffic yang menuju kepada *proxy* untuk diteruskan kepada *honeypot* yang berada pada sistem HaaS. Pembangunan *proxy* dilakukan dengan menggunakan *openvpn* sebagai penghubung jaringan, iptables rules untuk melakukan redireksi, dan programming socket untuk membuat *service* palsu agar dapat menjebak pengguna untuk melakukan serangan. Instalasi *openvpn* dilakukan dengan menggunakan package manager apt dengan menggunakan perintah $sudo apt install *openvpn* pengguna dapat memastikan apabila *openvpn* sudah ter-install dengan menjalankan perintah *openvpn*. Untuk dapat terhubung dengan sistem pengguna membutuhkan config file dengan ekstensi .ovpn yang didapatkan setelah pengguna melakukan registrasi. Apabila *proxy* dapat terhubung dengan sistem maka tahap selanjutnya adalah untuk mengaplikasikan rules iptables yang tercantum pada Gambar 20.

```
$ sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j DNAT --to-destination
192.168.42.21
$ sudo iptables -t nat -A POSTROUTING -p tcp -d 192.168.42.21 --dport 22 -j MASQUERADE
```

Gambar. 20 Perintah iptables pada *proxy*.

C. Availability Testing

TABEL VIIIII
PARAMETER AVAILABILITY TESTING

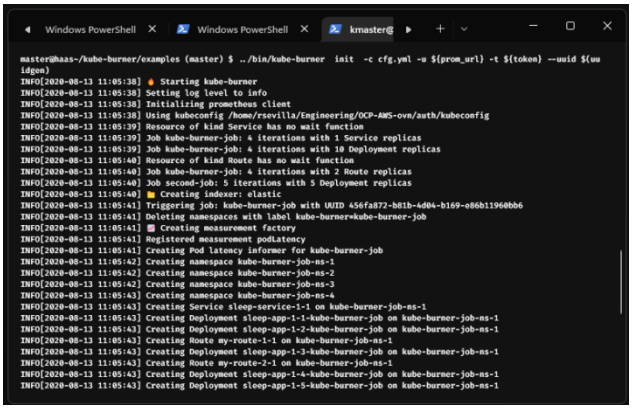
No	Parameter	plan	result	keterangan
Availability Test				
1	Apakah layanan dapat menyala kembali dengan waktu kurang dari 10s?	ya	ya	Ketersediaan layanan difasilitasi oleh Kubernetes *Cluster*
2	Apakah ketentuan pencadangan/kegaga lan/pemulihan bencana tersedia?	ya	ya	Migrasi dapat dengan mudah dilakukan dengan menambahkan *node* external dengan kubeadm
Failover Test				
3	Apakah terdapat perubahan pada konfigurasi layanan?	tidak	tidak	Ketika failover terjadi konfigurasi tidak terjadi perubahan
4	Apakah fungsi failover berjalan ketika terjadi kegagalan?	ya	ya	Implementasi replicaset untuk mengantisipasi layanan yang mengalami kegagalan
5	Apakah dalam waktu 60s sistem kembali normal?	tidak	tidak	Performa mengalami penurunan sesaat ketika terjadi failover pada salah satu *pod*
6	Apakah terdapat data yang hilang pada layanan?	tidak	tidak	*Pod* yang mengalami kerusakan tidak kehilangan data, karena data terpisah dari *pod*(terdapat pada volume)
7	Apakah terdapat pemantauan terhadap kegagalan atau kerusakan?	ya	ya	Terdapat log yang menunjukkan terjadinya kegagalan atau kerusakan

Hasil yang diharapkan pada Tabel 8 merupakan pengolahan dari analisis risiko pada indikator ketersediaan dan kontinuitas sistem [24]. Ketersediaan menjadi permasalahan ketika sumber daya pada sistem digunakan secara bersamaan antar pengguna. Hal ini dapat berdampak pada keberlangsungan sistem, ketika terdapat salah satu layanan yang menjadi target serangan penolakan layanan terdistribusi(DDoS), dapat menyebabkan layanan tersebut merusak kinerja dari layanan lainnya sehingga mengganggu pengguna lainnya [27]. Berdasarkan pemetaan pengujian pada Tabel 8, sistem *Honeypot*-as-a-*Service* mampu memberikan ketersediaan bagi pengguna. Dari tujuh kriteria yang diujikan, terdapat 1 kriteria yang tidak memenuhi namun tidak berdampak besar bagi keberlangsungan sistem.

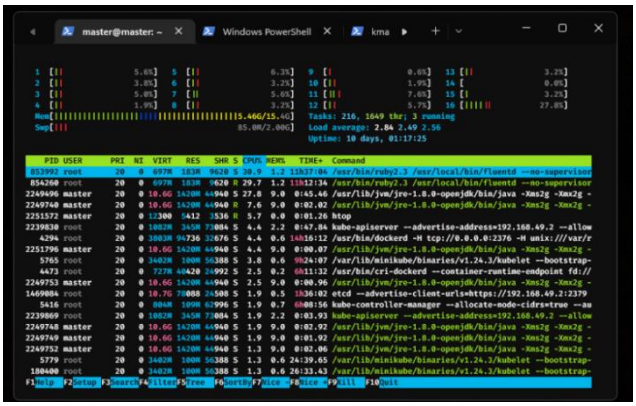
D. Performance Testing

1) Stress Testing: Pengujian dilakukan dengan menggunakan kube-burner yang merupakan *tools* yang didesain untuk melakukan *stress test* pada lingkungan kubernetes [28]. Gambar 21 menunjukkan *tools* kube-burner yang sedang melakukan *stress test* pada sistem HaaS.

Pengamatan pada tahap *stress* testing ini dilakukan dengan menggunakan *tools resource monitoring htop*. Pada Gambar 22 htop menampilkan *utilisasi* penggunaan CPU dengan keterangan pada masing-masing core dan htop juga menampilkan penggunaan Memory serta proses yang sedang berjalan.

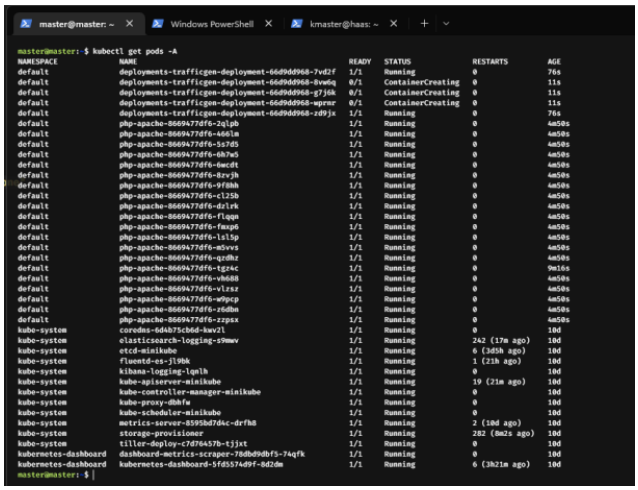


Gambar. 21 Stress Testing menggunakan kube-burner.

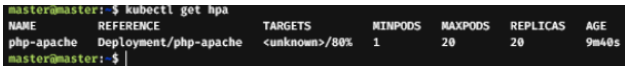


Gambar. 21 Resource monitoring dengan Htop.

2) *Load Testing:* Dalam lingkungan HaaS, pengujian ini dilakukan dengan membuat *deployment* web *service* berbasis PHP, yang kemudian akan dilakukan panggilan atau request dari *deployment* lainnya. Pada Gambar 21, terdapat *pod* berupa php-apache yang merupakan layanan web *service* . Layanan tersebut akan menerima request atau *query* dari *deployment* trafficgen. *Deployment* tersebut akan membuat lima *pod* dengan tujuan untuk merekayasa penggunaan akses terhadap web-*service* . Beban yang diberikan kepada sistem dalam tahap load test ini, akan digunakan untuk menguji skalabilitas dan kestabilan sistem dengan menggunakan fungsi Horizontal *Pod* Scaler(HPA) pada kubernetes. HPA akan mereplikasi layanan yang melebihi kapasitasnya dengan mengalokasikan *pod* replika baru untuk digunakan. Gambar 24 mendefinisikan HPA akan mereplika *pod* php-apache dengan maksimal parameter sebanyak 20 replika untuk masing-masing *pod*. Pada saat yang bersamaan dilakukan serangan *flooding* dengan menggunakan hping. Serangan tersebut akan memenuhi network interface pada sistem dengan paket data dalam jumlah yang besar. Banyaknya paket data yang masuk ke dalam sistem akan membebani kapasitas jaringan dan kemampuan untuk memproses informasi sehingga, akses pengguna ke dalam sistem akan terganggu.

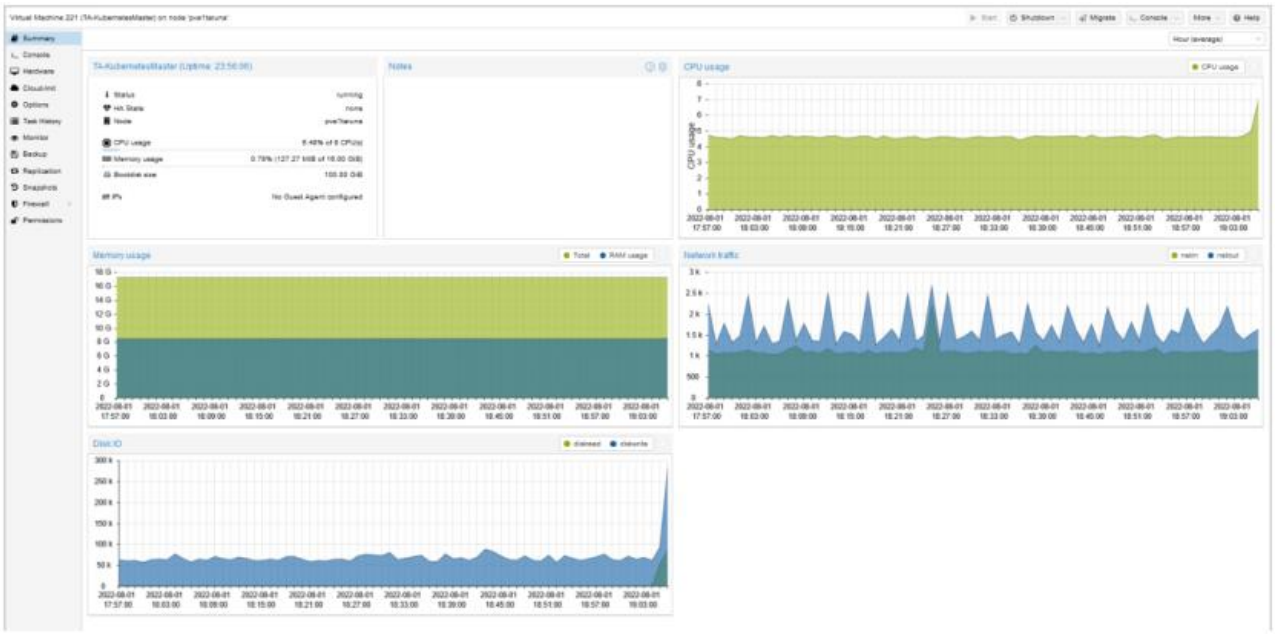


Gambar. 22 Load Testing pada HaaS.



Gambar. 23 AutoScaler untuk web*service* .

Pemantauan penggunaan daya secara eksternal turut dilakukan dengan menggunakan resource monitoring yang tertanam dalam Proxmox VE. Gambar 25 terdapat spike pada penggunaan CPU, I/O, dan Network ketika terjadi serangan flooding pada sistem.



Gambar. 24 Spike Pada Resource *Cluster*

Pengujian performa serta hasil yang diharapkan pada Tabel 9 diolah dari *checklist* kriteria dan risiko pada performa *Cloud Computing* [23]. Berdasarkan 10 kriteria yang telah ditetapkan terdapat 6 kriteria pada *stress and load test* dan 4 kriteria pada *Scalability Test*.

TABEL IX
PARAMETER PERFORMANCE TESTING

No	Parameter pengujian	Plan	result	keterangan
Stress & Load Test				
1	Apakah waktu respons dibawah 1 detik?	ya	ya	Waktu respons cukup, namun terdapat spike ketika terdapat layanan yang melebihi kapasitas pada kluster
2	Apakah kapasitas pemrosesan dengan 32 Core CPU memenuhi kebutuhan?	ya	ya	Pemrosesan sistem ketika dalam average load dan scenario test memenuhi kebutuhan dari sistem dimana, sistem masih dapat merespon *query* terhadap sistem tanpa delay
3	Apakah perangkat yang digunakan berpengaruh pada Performa sistem?	tidak	tidak	Perangkat tidak mempengaruhi performa, dikarenakan load pada sistem disebar keseluruh *node*
4	Apakah Performa bervariasi karena jaringan tidak stabil?	tidak	ya	Koneksi internet yang tidak mencukupi mengakibatkan kenaikan pada penggunaan resource
5	Apakah latency melebihi 100ms?	Tidak	tidak	Latency berada pada range 40-80ms
6	Apakah performa menurun dalam rentang waktu 1-5 menit?	Tidak	ya	Meningkatnya beban pada sistem ketika berjalan dalam waktu yang berkelanjutan
Scalability Test				
7	Apakah pelanggan lain mempengaruhi kinerja dari sistem?	tidak	iya	Terjadi Spike pada resource ketika terdapat pelanggan yang menggunakan layanan melebihi kapabilitasnya
8	Apakah sistem	ya	ya	Skalabilitas pada

No	Parameter pengujian	Plan	result	keterangan
	dapat diluaskan secara vertikal maupun horizontal ?			klusterdapat diperluas dengan menambahkan *node* melalui kubeadm
9	Apakah performa dar sistem dapat dikembangkan?	ya	ya	Performa sistem dapat berkembang dengan menambahkan *node* atau *cluster*
10	Apakah performa tidak mencukupi karena terjadi overbooking?	Tidak	tidak	Layanan tidak akan berjalan selama konfigurasi tidak diimplementasikan pada sistem

Dalam *Stress & Load Test*, parameter yang tidak memenuhi kriteria tidak tercapai dikarenakanan batasan perangkat penelitian yang digunakan. "*Bandwidth* yang tidak mencukupi" dikarenakan *resource* pada *Datacentre* terbagi dengan pengguna lainnya dan *bandwidth* itu sendiri terbatas dan tidak memungkinkan untuk dikembangkan pada saat ini. "Performa Internet mempengaruhi kinerja" merupakan anomali yang terjadi ketika terjadi serangan *flooding* pada jaringan, seharusnya internet tidak mengganggu atau berdampak pada sistem itu sendiri [27]. "Performa menurun seiring waktu" pada parameter ini, kinerja sistem mengalami penurunan ketika telah dilakukan beberapa kali pengujian, hal ini disebabkan karena tidak adanya penjadwalan untuk melakukan *maintenance* pada *node* [29]. Kemudian, pada *Scalability Test* parameter "pelanggan lain yang mempengaruhi kinerja" tidak terpenuhi dikarenakan batasan penggunaan sumber daya yang dibatasi terlalu besar, dikarenakan untuk *mengakomodir* apabila terjadinya *overhead* pada salah satu pengguna yang melebihi batasan kemampuan layanannya [19].

E. Functional Testing

TABEL X
HASIL FUNCTIONAL TESTING

No	Parameter pengujian	Skor	Keterangan
1	Apakah layanan dapat beradaptasi dengan lingkungan tertentu?	66%	Dikarenakan oleh homogenitas lingkungan yang digunakan oleh responden
2	Apakah terdapat banyak konfigurasi yang diperlukan?	54%	Masih bisa dilakukan simplifikasi terhadap penerapan dalam jaringan pengguna
3	Apakah kustomisasi mungkin?	54%	Kustomisasi memungkinkan namun terbatas
4	Apakah (banyak) perangkat lunak khusus tambahan diperlukan?	57%	dapat dilakukan simplifikasi terhadap penerapan dalam jaringan pengguna
5	Apakah platform yang diperlukan didukung?	66%	Platform yang digunakan adalah Linux
6	Apakah mungkin untuk menghubungkan/menginteg rasikan layanan dengan sistem lain?	54%	
7	Apakah tersedia cukup manual dan/atau kursus?	64%	Petunjuk yang diberikan mencukupi

Hasil yang tertera pada Tabel 10 menunjukkan bahwa skor yang didapat berada pada rentang diantara 50% dan 75%. Total skor yang didapat adalah 59,4% yang didapatkan dengan melakukan rata-rata nilai keseluruhan pertanyaan. Dengan didapatkannya skor tersebut menandakan bahwa fungsionalitas sistem masih memiliki banyak kekurangan karena hanya bisa mendapatkan skor 59,4% dari target pencapaian 75%.

F. Scenario Testing

TABEL XI
HASIL KUESIONER SCENARIO TESTING

No	Pertanyaan	Skor
1	Apakah menurut anda PC 1 (.170) merupakan sebuah *Honeypot*?	55%
2	Apakah menurut anda PC 2 (.156) merupakan sebuah *Honeypot*?	48%
3	Apakah menurut anda PC 3 (.191) merupakan sebuah *Honeypot*?	52%

Pada Tabel 11 Lebih dari 50% responden dapat mengidentifikasikan PC 1 sebagai sebuah *honeypot*, hal ini didukung dari fakta bahwa *honeypot* tersebut merupakan *honeypot* interaksi rendah. Kurang dari 48 % responden dapat mengidentifikasi PC 2 sebagai *honeypot*, dalam hal ini yaitu *honeypot* yang terhubung dengan sistem. Kemudian, 52% responden salah mengidentifikasi PC 3 yang merupakan sebuah Server berbasis Ubuntu sebagai *Honeypot*. Maka, dapat disimpulkan bahwa *Honeypot* yang berada pada sistem belum bisa memenuhi target yaitu, dimana 2/3 responden tidak dapat mengidentifikasikan *honeypot* yang terhubung pada sistem.

V. KESIMPULAN

Honeypot-as-a-*Service* dibangun dengan orkestrasi Kubernetes *Cluster* dapat memberikan ketersediaan dan performa yang menunjang kebutuhan dari pengguna. Dalam hal terjadi kerusakan maupun kegagalan pada sistem maupun layanan, fungsi Failover dan Load Balancer berhasil untuk menangani kegagalan tersebut. Kegagalan yang terjadi, tidak terlalu berdampak pada keseluruhan kinerja sistem namun, dapat menimbulkan spike dalam penggunaan sumber daya. Berdasarkan pengujian yang dilakukan dengan responden, Sistem *Honeypot*-as-a-*Service* ini belum memenuhi kriteria dari pengguna. Dari hasil pengujian Fungsional dan Pengujian User, didapatkan nilai yang masih belum memuaskan. Diperlukan pengembangan lebih lanjut terhadap aksesibilitas sistem dan layanan yang nantinya ditawarkan kepada pengguna. Aksesibilitas merupakan hal yang penting, dikarenakan merupakan salah satu indikator utama dalam *Cloud Computing*. *Honeypot* belum efektif dalam menjebak penyerang, dilihat dari scenario test yang dilakukan oleh 20 orang responden dapat mengidentifikasi layanan *honeypot* sebagai *honeypot* yang sebenarnya sehhingga perlu dikembangkan dengan *penggunan high level interaction honeypot*.

ACKNOWLEDGMENT

Terima kasih kepada Politeknik Siber dan Siber Negara sebagai penyandang dana *submission* pada jurnal terakreditasi ini.

REFERENSI

[1] M. Cusumano, "*Cloud Computing* and SaaS as New Computing Platforms," *Commun. ACM*, vol. 53, no. 4, pp. 27–29, Apr. 2010,

[2] "Study: Hackers Attack Every 39 Seconds | A. James Clark School of Engineering, University of Maryland." https://eng.umd.edu/news/story/study-hackers-attack-every-39-seconds (accessed Jan. 25, 2023).

[3] Michael J. Kavis, *Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS)*, 1st Editio. Wiley, 2014.

[4] J. M. Jorquera Valero, M. Pérez, A. Huertas, and G. Martinez Perez, "Identification and classification of cyber threats through SSH honeypot systems," in *Handbook of Research on Intrusion Detection Systems*, IGI Global, 2020, pp. 105–129.

[5] R. M. Campbell, K. Padayachee, and T. Masombuka, "A survey of honeypot research: Trends and opportunities," in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2015, pp. 208–212.

[6] A. Tiwari and D. Kumar, "Comparitive Study of Various Honeypot Tools on the Basis of Their Classification & Features," *SSRN Electron. J.*, 2020,

[7] J. H. Jafarian and A. Niakanlahiji, "Delivering honeypots as a *service* ," *Proceedings of the Annual Hawaii International Conference on System Sciences*, vol. 2020-Janua. pp. 1835–1844, 2020.

[8] M. Boorshtein and S. Surovich, *Kubernetes and Docker - An Enterprise Guide: Effectively containerize applications, integrate enterprise systems, and scale applications in your enterprise*, 2nd Editio. Birmingham: Packt Publishing, 2021.

[9] G. N. Schenker, H. Saito, H.-C. C. Lee, and H. K.-J. Carol, *Getting Started with Containerization: Reduce the operational burden on your system by automating and managing your containers*. Birmingham: Packt Publishing, 2019.

[10] V. Sharma, *Beginning Elastic Stack*, 1st ed. Berkeley, CA: Apress, 2016.

[11] S. Chhajed, *Learning ELK Stack*. Birmingham: Packt Publishing, 2015.

[12] A. Elazazy, "Honey*Proxy* Implementation in Cloud Environment with Docker *Container* HoneyFarm," 2018, [Online]. Available: https://web-*proxy*.io/*proxy*/dspace.ut.ee/handle/10062/66115

[13] A. Lysholm and J. Valfridsson, "Developing a modular , high-interaction honeypot using *container* virtualization," 2021.

[14] J. Buzzio-Garcia, "Creation of a High-Interaction Honeypot System based-on Docker *container*s," in *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, 2021, pp. 146–151.

[15] D. Sever and T. Kišasondi, "Efficiency and security of docker based honeypot systems," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 1167–1173.

[16] J. vom Brocke, A. Hevner, and A. Maedche, "Introduction to Design Science Research," in *Design Science Research Cases*, J. vom Brocke, A. Hevner, and A. Maedche, Eds. Cham: Springer International Publishing, 2020, pp. 1–13.

[17] V. K. Vaishnavi and W. Kuechler, *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*, 2nd ed. Florida: CRC Press, 2015.

[18] C. Gupta, "HoneyKube : designing a honeypot using micro*service* s-based architecture." Aug. 30, 2021. [Online]. Available: http://essay.utwente.nl/88323/

[19] The Linux Foundation, "Kubernetes." https://kubernetes.io/ (accessed Jan. 26, 2023).

[20] Y.-L. Chen and Q. Li, *Modeling and Analysis of Enterprise and Information Systems*. Berlin: Springer Berlin Heidelberg, 2009.

[21] G. Sayfan, *Mastering Kubernetes: Large scale container deployment and management*. Birmingham: Packt Publishing, 2017.

[22] L. T. M. Blessing and A. Chakrabarti, *DRM, a Design Research Methodology*. London: Springer London, 2009.

[23] K. Blokland, J. Mengerink, and M. Pol, *Testing Cloud Service s: How to Test SaaS, PaaS & IaaS*, 1st ed. San Rafael: Rocky Nook, 2013.

[24] I. M. Al Jawarneh *et al.*, "*Container* Orchestration Engines: A Thorough Functional and Performance Comparison," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[25] 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, pp. 74–78.

[26] A. Yahyaoui, "Testing Deceptive Honeypots," Naval Postgraduate School, 2014.

[27] E. Bauer and R. Adams, *Reliability and Availability of Cloud Computing*. Hoboken, New Jersey: Wiley-IEEE Press, 2012.

[28] "Develop, deploy, and evolve with Red Hat Hybrid Cloud | Red Hat Hybrid Cloud." https://cloud.redhat.com/ (accessed Jan. 26, 2023).

[29] J. Shuja, K. Bilal, S. A. Madani, and S. U. Khan, "Data center energy efficient resource scheduling," *Cluster Comput.*, vol. 17, no. 4, pp. 1265–1277, 2014.