Membangun Infrastruktur Untuk Kebutuhan Web Otomasi Multivendor Dan *Monitoring* Pada Sistem Arsitektur *Microservice*

1st Dharma Agung Sitepu
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
dharmaagung@student.telkomuniversity.ac.id

2nd Favian Dewanta
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
favian@telkomuniversity.ac.id

3rd Bagus Aditya
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
goesaditya@telkomuniversity.ac.id

Abstrak* — Dalam era digital yang terus berkembang, kebutuhan akan otomasi jaringan dan *monitoring* menjadi semakin penting, terutama dalam lingkungan yang menggunakan berbagai perangkat dari ve-ndor yang berbeda. Kompleksitas yang timbul dari integrasi perangkat multivendor sering kali menghambat efisiensi operasional dan menimbulkan tantangan besar dalam penerapan otomasi jaringan. Dalam konteks arsitektur *microservice*, tantangan ini semakin diperparah oleh kebutuhan kompabilitas dan fleksibilitas antar layanan yang berjalan secara terpisah. Penelitian ini bertujuan untuk merancang dan mengimplementasikan infrastruktur web yang dapat mendukung otomasi multivendor dan *monitoring* secara efektif pada sistem berbasis arsitektur *microservice*. Dengan memanfaatkan protokol *Secure Shell* (SSH) dan *Simple Network Management Protocol* (SNMP), sistem ini akan menyediakan platform yang terintegrasi untuk *provisioning* dan *monitoring* perangkat jaringan dari berbagai vendor. Sistem ini juga akan memanfaatkan teknologi kontainer dan orkestrator seperti Kubernetes untuk memastikan skalabilitas dan kelangsungan layanan. Platform ini diharapkan mampu memberikan solusi yang efisien untuk menangani berbagai tantangan dalam integrasi perangkat multivendor serta mendukung pengembangan lebih lanjut dari sistem otomasi jaringan berbasis web. Penelitian ini juga akan melakukan evaluasi penggunaan berbagai alat otomasi dan *monitoring* yang sudah ada, serta memperkenalkan konsep *microservice* sebagai pendekatan yang lebih fleksibel dan scalable untuk pengelolaan jaringan. Sistem yang telah kami rancang mampu melakukan otomasi pada 10 perangkat secara bersamaan dengan waktu 8 detik dan untuk monitoring dibutuhkan waktu 4 detik.

Kata kunci -- ***Otomasi Jaringan, Multivendor, Arsitektur Microservice, Monitoring, Provisioning, SSH, SNMP***

I. PENDAHULUAN

Vendor dalam industri jaringan bertindak sebagai penyedia perangkat keras dan lunak yang krusial untuk mendukung infrastruktur jaringan perusahaan, termasuk *Internet Service Provider* (ISP). Seiring dengan berkembangnya kebutuhan perusahaan, ISP sering kali harus mengintegrasikan perangkat dari berbagai vendor, yang dapat menimbulkan tantangan tersendiri, terutama dalam hal otomasi jaringan [1]. Setiap vendor memiliki alat dan protokol yang berbeda, sehingga menyulitkan integrasi dan otomasi dalam lingkungan yang heterogen. Kompleksitas ini semakin terasa dalam konteks arsitektur *microservice*, di mana layanan terpisah harus dapat bekerja secara kompatibel dan efisien. Salah satu tantangan utama adalah integrasi SNMP dengan sistem otomasi, di mana data sensor dari SNMP dapat digunakan sebagai pemicu untuk menjalankan skrip otomasi [2]. Penelitian sebelumnya menunjukkan bahwa Python, dengan library seperti Paramiko dan Netmiko, telah berhasil diterapkan untuk otomasi konfigurasi perangkat jaringan dari berbagai vendor menggunakan protokol SSH. Protokol SSH terbukti efektif dalam menyederhanakan proses *provisioning* dan konfigurasi secara masal.

Alat otomasi seperti Ansible Tower juga populer digunakan untuk mengelola perangkat dan layanan dalam lingkungan multivendor [3]. Penelitian ini akan mengembangkan infrastruktur berbasis web yang mendukung otomasi multivendor dan *monitoring* dalam arsitektur *microservice*. Dengan menggabungkan protokol SSH dan SNMP, serta memanfaatkan kontainer dan orkestator seperti Kubernetes, penelitian ini bertujuan untuk mengatasi tantangan integrasi multivendor dan memberikan solusi yang skalabel dan efisien [4]. Sebagai emulator untuk mempermudah pengembangan terhadap perangkat multivendor penelitian ini menggunakan software emulator Pnetlab yang dapat menjalankan banyak perangkat vendor [5].

II. KAJIAN TEORI

A. Arsitektur *Microservice* dalam Otomasi Jaringan

Arsitektur *microservice* merupakan pendekatan yang membagi sistem menjadi beberapa layanan kecil yang dapat dikelola dan dikembangkan secara independen [6]. Dalam konteks otomasi jaringan, *microservice* memungkinkan setiap layanan otomasi dan *monitoring* dikelola secara modular, meningkatkan fleksibilitas dan skalabilitas. Dengan arsitektur ini, perubahan atau penambahan layanan dapat dilakukan tanpa mengganggu keseluruhan sistem, yang sangat berguna dalam lingkungan multivendor yang dinamis.

B. Kontainerisasi dan Orkestrasi dengan Docker dan Kubernetes

Kontainerisasi menggunakan Docker memungkinkan setiap layanan *microservice* dijalankan dalam lingkungan

yang terisolasi [7]. Kubernetes sebagai orkestator mengelola kontainer ini, memastikan bahwa aplikasi berjalan dengan reliabilitas dan efisiensi yang tinggi. Dalam arsitektur ini, Kubernetes menyediakan kemampuan otomatis untuk *scaling* dan *load balancing*, yang penting untuk menangani beban kerja otomasi jaringan secara dinamis.

C. Protokol SSH dan SNMP untuk Otomasi Jaringan

SSH dan SNMP adalah dua protokol kunci dalam otomasi jaringan. SSH digunakan untuk mengelola perangkat jaringan secara aman melalui antarmuka baris perintah, sementara SNMP memungkinkan pengumpulan data jaringan dan pemantauan kondisi perangkat. Dalam sistem yang diusulkan, kombinasi kedua protokol ini memungkinkan *provisioning* dan *monitoring* perangkat jaringan dari berbagai vendor dengan efisien.

D. Alat Otomasi dan *Monitoring*

Alat otomasi yang dibuat mengunakan bahasa pemrograman Python, dan alat *monitoring* seperti Prometheus dan Grafana, menjadi komponen penting dalam ekosistem otomasi jaringan. Paramiko yang merupakan library pada python memudahkan konfigurasi perangkat secara otomatis melalui protokol SSH, sementara Prometheus dan Grafana digunakan untuk memantau performa dan status jaringan. Integrasi alat ini dalam arsitektur *microservice* memberikan visibilitas dan kontrol yang lebih baik terhadap infrastruktur jaringan.

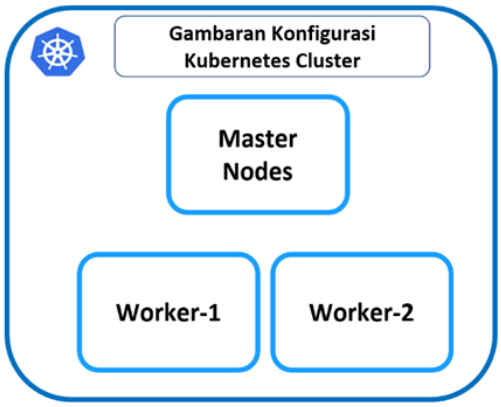
E. Pnetlab

Pnetlab adalah *software* emulasi open-source yang akan digunakan pada penelitian kali ini. *Software* emulasi ini dapat menjalankan sistem operasi *router* dan server berbagai vendor, hal ini dapat mempermudah penelitian dan pengujian untuk melihat performa otomasi pada beragam perangkat dan banyak perangkat sekaligus, dengan menggunakan Pnetlab pengujian perangkat bisa dilakukan dengan lebih dinamis.

III. METODE

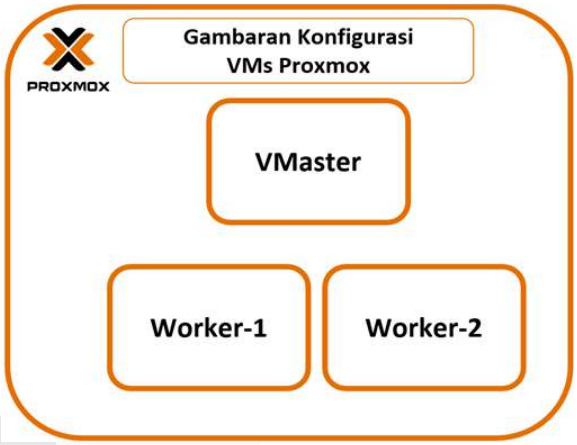
A. Konsep Arsitektur

Konsep arsitektur yang diusulkan dalam penelitian ini berpusat pada penggunaan arsitektur *microservice* untuk mendukung otomasi *provisioning* dan *monitoring* perangkat jaringan multivendor. Arsitektur ini terdiri dari beberapa komponen yang bekerja secara terpisah namun saling terintegrasi melalui Application Programming Interface (API).



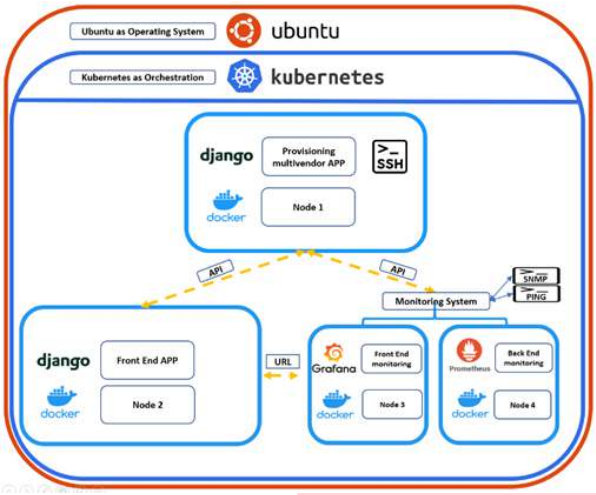
GAMBAR 1
Konfigurasi *nodes* pada Kubernetes

Gambar 1 menjelaskan konfigurasi dan *nodes* yang akan digunakan pada pembentukan arsitektur *microservice*, setiap layanan dalam sistem ini menggunakan Docker sebagai sistem kontainer, sementara Kubernetes berperan sebagai orkestrator untuk mengelola *deployment* dan *scaling*. Proxmox digunakan sebagai platform virtualisasi untuk menjalankan virtual machines (VMs) untuk kebutuhan Kubernetes *cluster*. Gambar 2 memperlihatkan bagaimana konfigurasi VMs pada proxmox akan dibuat sama dengan konfigurasi pada Kubernetes *cluster* [8].



GAMBAR 2
Konfigurasi VMs pada Proxmox

VMs *nodes* pada server Proxmox diatur sebagai *cluster* Kubernetes yang terdiri dari 1 master *nodes* dan 2 *worker nodes*. *Cluster* tersebut memastikan bahwa sistem dapat menangani peningkatan beban kerja dan *high availability* dengan menambahkan atau mengurangi *container* sesuai kebutuhan [9]. Untuk memastikan kompabilitas dan fleksibilitas, setelah sistem otomasi dan *monitoring* dijalankan pada arsitektur *microservice* yang telah dibuat, Pnetlab sebagai emulator perangkat jaringan digunakan untuk membantu menguji kompatibilitas dan performa otomasi terhadap berbagai perangkat multivendor.



GAMBAR 3
Konfigurasi Keseluruhan Sistem

Gambar 3 menjelaskan bagaimana sistem *provisioning* dan *monitoring* otomatis perangkat jaringan multivendor pada arsitektur *microservice* yang saling bekerjasama menggunakan 4 *nodes* Docker *container*. Docker dipilih karena akan tercapai efektifitas dan efisiensi kerja mesin karena sumber daya mesin akan digunakan secara maksimal untuk menjalankan layanan yang ada berbasis virtual [10].

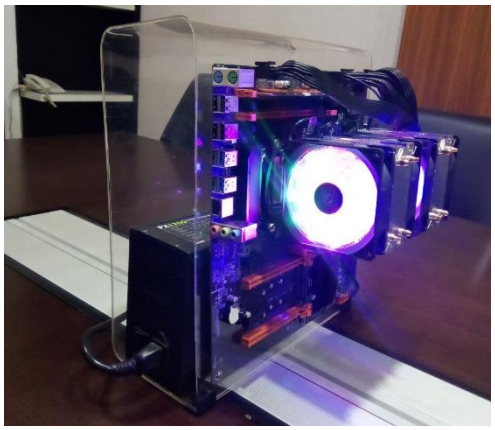
Nodes 1 menjalankan Sistem back-end *provisioning* yang terhubung ke perangkat jaringan menggunakan SSH, *nodes* 2 sistem front-end *provisioning* yang terhubung ke back-end menggunakan API, *nodes* 3 dan 4 menjalankan sistem *monitoring* menggunakan Grafana dan Prometheus yang terhubung ke sistem back-end *provisioning* menggunakan API dan terhubung ke sistem front-end *provisioning* menggunakan *Uniform Resource Locator* (URL). Arsitektur *microservice* digunakan karena terdapat limitasi pada sistem *monitoring* jika perangkat jaringan sudah melebihi kapasitas *resource* sistem *monitoring*, arsitektur *microservice* memungkinkan sistem *monitoring* dan sistem lainnya dapat melakukan *scale up* secara horizontal tanpa membuat *host* VMs baru.

B. Implementasi

1. Persiapan Perangkat Keras

Persiapan dan konfigurasi perangkat keras adalah hal yang pertama dilakukan dalam implementasi sistem yang akan dibuat. Gambar 4 menampilkan komputer sebagai server Proxmox yang akan digunakan dengan spesifikasi komputer sebagai berikut:

- Intel Xeon V3 36 *Core* 72 *Thread*
- RAM 64GB DDR4 ECC
- SSD NVMe 512GB



GAMBAR 4
Komputer *Server*

Router diperlukan untuk melakukan perutean ke *Internet Protocol* (IP) *Address Public* dan melakukan *load balance* ke setiap *nodes cluster* Kubernetes. secara umum, semua *router* dari berbagai macam vendor dapat digunakan seperti ditunjukkan pada referensi [11]. Gambar 5 merupakan perangkat *router* Mikrotik yang digunakan dengan seri CCR1009-7G-1C-PC:

GAMBAR 5
Router Mikrotik

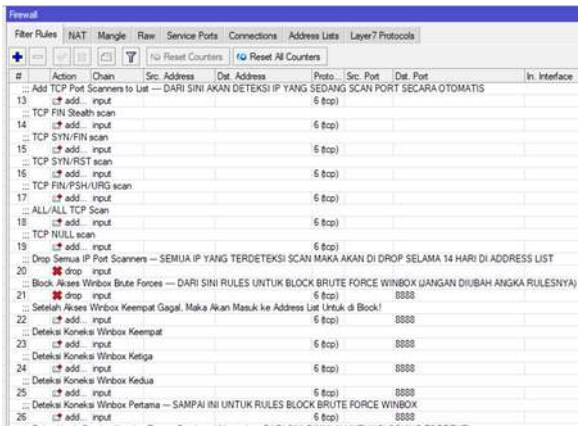
Setelah semua *hardware* menyala, kami melakukan instalasi sistem operasi ubuntu untuk kebutuhan berikut:

1. VMs Pnet-Lab untuk melakukan vitualisasi perangkat jaringan yang akan diuji coba sebagai target
2. VMs Master sebagai controller dari Kubernetes *cluster*
3. VMs Worker-1 dan Worker-2 sebagai mesin kerja dari Docker yang sudah dilakukan *deployment* dan di konfigurasi pada VM Master

Gambar 6 menunjukkan VMs yang sedang berjalan pada Proxmox. Spesifikasi setiap VMs yang digunakan disesuaikan dengan kebutuhan optimum aplikasi yang dijalankan. Adapun konfigurasi pada router untuk mendukung VMs sebagai berikut:

GAMBAR 6
VMs Berjalan pada Proxmox

a. Selanjutnya konfigurasi *router* untuk keperluan *firewall*, *forward port* dan domain ke *provider*, bisa dilihat pada Gambar 7 konfigurasi pada *router* untuk *port forwarding* ke domain dan IP publik.

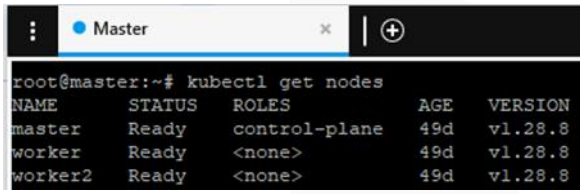


GAMBAR 7
Konfigurasi *Router*

b. Konfigurasi *firewall router* untuk mendeteksi *scanner* dan serangan dari luar, agar IP publik *router* tersebut terlindungi dari serangan.
c. Selanjutnya lakukan konfigurasi *forward* domain dan port untuk kebutuhan akses *website* dan sistem lainnya dari internet dan domain allspide.com.

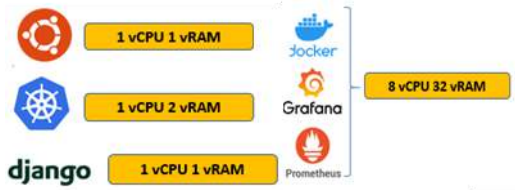
2. Instalasi Kubernetes dan Deployment software

Nodes pada Kubernetes *cluster* yang telah aktif dapat dilihat pada Gambar 8 di mana terdapat 1 *nodes* master dan 2 *nodes worker* sesuai pada konsep rancangan.



GAMBAR 8
Kubernetes *Cluster Nodes*

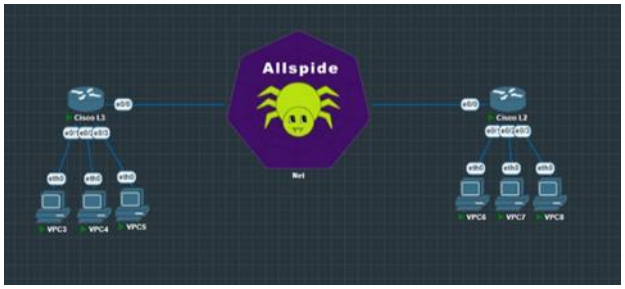
Deployment untuk aplikasi yang dibutuhkan dalam pengembangan *website* dilakukan pada VMs Master. Gambar 9 memperlihatkan bagaimana membuat *deployment* aplikasi menggunakan *file* berformat *Yet Another Markup Language* (YAML). *File* tersebut diberikan nama *deployment* "prometheus-config" menggunakan image Prometheus dengan *job* "Prometheus" dan "SNMP".



GAMBAR 9
Deployment Prometheus dengan YAML

3. Installasi Pnetlab

Image Pnetlab dapat diunduh pada *website* pnetlab.com. Selanjutnya image tersebut dapat dijalankan pada Proxmox secara langsung. Penelitian ini telah menjalankan beberapa perangkat jaringan dari vendor yang berbeda seperti Cisco, Juniper, dan Mikrotik. Perangkat yang telah diemulasi ini nantinya terhubung ke jaringan *cluster* Proxmox seperti pada Gambar 10, agar dapat berkomunikasi dengan jaringan lokal *website* melalui *router*.



GAMBAR 10
Konfigurasi Pnetlab

IV. HASIL DAN PEMBAHASAN

Sistem yang dibuat membutuhkan 4 *deployment* aplikasi yang akan saling bekerjasama untuk menghasilkan fitur otomasi konfigurasi perangkat multivendor dan *monitoring*. Gambar 11 menunjukkan aplikasi yang berjalan pada Kubernetes *cluster*. Aplikasi yang telah berhasil dijalankan antara lain:

- Prometheus
- Grafana
- Django Framework
- SNMP Exporter

```
root@master:~# ls
django.yaml  grafana.yaml  prometheus.yaml  snmp-exporter.yaml
root@master:~# kubectl get pods
NAME                                    READY   STATUS
RESTARTS   AGE
django-app-655f66d48b-plh8k             0/1     Running            1
grafana-deployment-589c8b49d8-2fn4p     1/1     Running            3
prometheus-6f749f667c-rcvf6             1/1     Running            0
snmp-exporter-b464b99c9-s91ln           1/1     Running            3
```

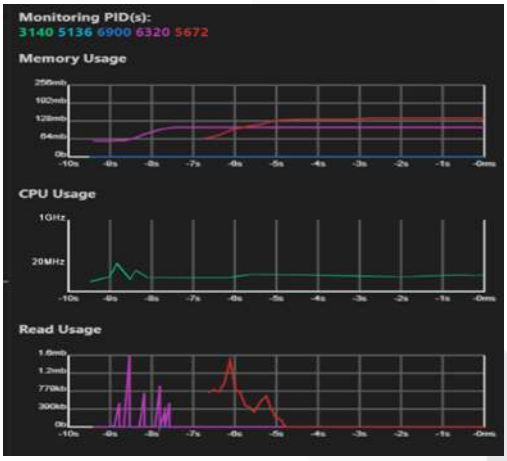
GAMBAR 11
Deployment pada Kubernetes

Setelah semua *deployment* aplikasi selesai dilakukan dan sistem web dimuat ke dalam infrastruktur *microservice, resource hardware* yang diperlukan adalah 11 Core vCPU 35 GB vRAM. Spesifikasi tersebut adalah spesifikasi optimum yang sudah di ukur untuk menjalankan sistem *provisioning* dan *monitoring* otomatis perangkat jaringan multivendor pada arsitektur *microservice* secara optimal. Detail pembagian komponen *software* yang menggunakan resource dapat dilihat pada Gambar 12.

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      evaluation_interval: 15s
    scrape_configs:
      - job_name: 'prometheus'
        static_configs:
          - targets: ['localhost:9090']
      - job_name: 'snmp'
        static_configs:
          - targets:
            - 139.255.41.67
        metrics_path: /snmp
```

GAMBAR 12
Total *Resource Hardware*

Pengujian yang dilakukan pada setiap fitur sudah sesuai dengan yang diharapkan. Ketika *website* dijalankan dan melakukan *provisioning* terhadap 10 perangkat secara bersamaan, sumber daya yang digunakan dapat dilihat pada Gambar 13. Adapun penggunaan RAM sebesar 128MB, penggunaan CPU pada 1 *core* sebesar 20 MHz, dan penggunaan penyimpanan sebesar 1.6MB.



GAMBAR 13
Grafik Hasil Pengujian

Untuk melakukan *provisioning* 10 perangkat secara bersamaan hanya dibutuhkan waktu paling lama 8 detik dan untuk *monitoring* dibutuhkan waktu 4 detik mulai dari *user* melakukan *submit* sampai mendapatkan respon balik dari setiap perangkat.

Solusi yang dibuat sudah menjawab permasalahan yang sebelumnya dirumuskan. faktor pendukung keberhasilan sistem yang dibuat adalah desain infrastruktur *microservice,* yang dapat melakukan *scaling resource* sesuai dengan kebutuhan serta penggunaan program Python yang mendukung SSH multivendor. Penerapan program basis kelas pada *framework* Django dapat melakukan pemrosesan paralel sehingga waktu yang dibutuhkan untuk melakukan otomasi dan *monitoring* lebih cepat. Sistem yang telah dibuat memiliki *high availability* 99,9% dengan bantuan fitur *self healing* yang digunakan pada Kubernetes *cluster.* Desain dan implementasi solusi ini memiliki keterbatasan dalam pengukuran terhadap keberagaman perangkat multivendor dan jumlah target perangkat dikarenakan keterbatasan perangkat yang dimiliki saat pengujian. Diharapkan untuk pengembangan lebih lanjut keberagaman target perangkat multivendor dan jumlahnya dapat ditingkatkan hingga kapasitas akhir 1 *nodes* dari Kubernetes *cluster,* agar dapat melakukan simulasi fitur *scale-up* yang dimiliki Kubernetes.

V. KESIMPULAN

Penelitian ini berhasil merancang dan mengimplementasikan infrastruktur berbasis *microservice* yang mendukung otomasi *provisioning* dan *monitoring* perangkat jaringan multivendor. Dengan menggunakan kombinasi teknologi kontainerisasi Docker dan orkestrasi Kubernetes, sistem yang dibangun mampu menyediakan solusi yang skalabel dan efisien untuk menangani kompleksitas integrasi perangkat dari berbagai vendor. Penggunaan protokol SSH dan SNMP memungkinkan integrasi otomasi dengan berbagai perangkat jaringan. Sementara pemanfaatan alat *monitoring* seperti Prometheus dan Grafana meningkatkan visibilitas dan kontrol terhadap sistem.

Pengujian menunjukkan bahwa sistem ini mampu melakukan *provisioning* dan *monitoring* secara cepat dan efisien, bahkan ketika menangani beberapa perangkat secara bersamaan. Meskipun demikian, penelitian ini memiliki keterbatasan dalam hal jumlah dan keberagaman perangkat yang diuji, area potensial untuk pengembangan lebih lanjut adalah integrasi yang lebih luas dengan perangkat multivendor dan optimalisasi *resource* pada cluster Kubernetes dapat menjadi fokus penelitian berikutnya.

REFERENSI

[1] J. G. Stevenson, "Management of multivendor networks," in IBM Systems Journal, vol. 31, no. 2, pp. 189-205, 1992.

[2] P. Roquero and J. Aracil, "On Performance and Scalability of Cost-Effective SNMP Managers for Large-Scale Polling," in IEEE Access, vol. 9,no. 1, pp. 7374-7383, 2021.

[3] A. Seck, C. S. E. Bassene, S. E. Zabolo and S. Ouya, "Building an interactive Software Defined Network from the MPSI for MPLS Service provisioning with Gitlab and Ansible," 2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Maldives, Maldives, pp. 1-6, 2022.

[4] O. Mart, C. Negru, F. Pop and A. Castiglione, "Observability in Kubernetes Cluster: Automatic Anomalies Detection using Prometheus," 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Yanuca Island, Cuvu, Fiji, pp. 565-570, 2020.

[5] A. A. Mazin, H. Z. Abidin, L. Mazalan and A. M. Mazin, "Network Automation Using Python

Programming to Interact with Multiple Third-Party Network Devices," 2023 10th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), Semarang, Indonesia, pp. 59-64, 2023.

[6] G. Blinowski, A. Ojdowska and A. Przybyłek, "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation," in IEEE Access, vol. 10, pp. 20357-20374, 2022.

[7] S. Han, K. S. Kim, and K. W. Rim, "Performance Analysis of Docker Container for Proxmox Virtual Environment," Proceedings of the 2019 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South, pp. 653-656), 2019.

[8] A. Rajković and M. Antić, "An environment for orchestrating containers on a local ProxMox server," 2024 Zooming Innovation in Consumer Technologies Conference (ZINC), Novi Sad, Serbia, pp. 179-181, 2024.

[9] B. F. Mubina, F. Dewanta and B. Aditya, "Performance Analysis of High Availability Video Conference Service with Kubernetes Cluster Across Data Center." 2022 2nd International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS), Yogyakarta, Indonesia, pp. 164-168, 2022.

[10] N. Zhou, H. Zhou and D. Hoppe, "Containerization for High Performance Computing Systems: Survey and Prospects," in IEEE Transactions on Software Engineering, vol. 49, no. 4, pp. 2722-2740, 1 April 2023.

[11] J. D. Morillo Reina and T. d. J. Mateo Sanguino, "Portable Device for Easy Management and Automatic Recovery of Networking Systems," in IEEE Latin America Transactions, vol. 17, no. 03, pp. 401-408, March 2019.