p-ISSN : 2406-7733
e-ISSN : 2597-9922

IMPLEMENTASI VIRTUALISASI SERVER BERBASIS DOCKER CONTAINER

Saleh Dwiyatno[1], Edy Rakhmat[2], Oki Gustiawan[3]
[1,3]Program Studi Sistem Komputer Fakultas Teknologi Informasi Universitas Serang Raya
[2]Program Studi Teknik Informatika Fakultas Ilmu Komputer Universitas Banten Jaya
salehdwiyatno@gmail.com[1], edyrakhmat@unbaja.ac.id[2], okigustiawan@gmail.com[3]

Abstrak **–** *Docker* adalah sebuah aplikasi yang berbasiskan teknologi *open source* yang memungkinkan *developer* atau siapapun untuk membuat, menjalankan, melakukan percobaan dan meluncurkan aplikasi di dalam sebuah *container*. *Docker* membuat proses pemaketan aplikasi bersama komponennya secara cepat dalam sebuah *container* yang terisolasi, sehingga dapat dijalankan dalam infrastruktur lokal tanpa melakukan perbuahan konfigurasi pada *container*. *Docker* juga sangat ringan dan cepat jika dibandingkan dengan mesin virtual yang berbasis *hypervisor*. SMK Negeri 1 Rangkasbitung mempunyai beberapa *server* yang mana didalamnya sudah menampung beberapa aplikasi *web*. *Server* tersebut berbasis *hypervisor* sehingga membutuhkan *resource* besar. Karena setiap VM menjalankan *guest* OS beserta kernelnya sendiri terpisah dari *host*. Oleh karena itu, dilakukan implementasi virtualisasi berbasis *docker container* supaya dapat meningkatkan efektifitas dalam penggunaan sumber daya CPU dan memori pada *server*. Pengujian dilakukan dengan cara melakukan variasi banyaknya jumlah *user request* yang berbeda pada masing-masing *container* menggunakan perangkat lunak *apache jmeter*.

Kata Kunci: *Container*, CPU, *Docker*, *Hypervisor*, Memori

I. PENDAHULUAN

Seiring dengan perkembangan komputer dan internet saat ini aplikasi berbasis *web* berkembang sangat pesat. Selain itu, aplikasi berbasis *web* juga semakin banyak digunakan karena dapat diakses di berbagai *platform* komputer hanya dengan menjalankan *web browser*. Sehingga, kemudahan proses *deployment* (penyebaran) aplikasi *web* beserta *software* pendukung seperti *web server*, *database server*, *dependency* dan *environment* lain ke *server* sangat dibutuhkan.

Secara umum ada dua metode *deployment* aplikasi *web* ke dalam *server*. Pertama menginstall aplikasi *web* beserta *environment* yang dibutuhkan ke dalam server tunggal, kelebihannya adalah melakukan konfigurasi terhadap *server* lebih mudah, sederhana dan cepat dalam proses *deployment*. Tetapi metode tersebut memiliki kekurangan yaitu setiap aplikasi tidak terisolasi, sehingga apabila mendeploy beberapa aplikasi yang masing-masing memiliki ketergantungan dengan paket versi tertentu maka dapat menimbulkan *Dependecie Hell*.

Metode yang kedua yaitu dengan memanfaatkan teknologi virtualisasi berbasis *hypervisor*, jadi setiap aplikasi dan *dependency* yang dibutuhkan di-*deploy* ke dalam mesin *virtual* yang berbeda. Dengan metode ini dapat meningkatkan skalabilitas, karena setiap aplikasi berjalan pada sumber daya (CPU, memori, penyimpanan data) yang berbeda sehingga dapat dengan mudah ditambahkan sesuai kebutuhan. Berdasarkan pada pengamatan yang telah dilakukan oleh penulis di ICT SMK Negeri 1 Rangkasbitung terdapat beberapa *server* yang mana di dalamnya sudah menampung beberapa aplikasi *web*. *Server* tersebut berbasis *hypervisor* sehingga membutuhkan *resource* besar. Karena setiap VM menjalankan *guest OS* beserta kernelnya sendiri terpisah dari *host*.

Docker sangat ringan dan mempunyai mekanisme yang lebih maju jika dibandingkan dengan perangkat lunak virtualisasi berbasis *hypervisor*. Indikasinya adalah adanya efektivitas lebih pada Docker dalam hal penggunaan sumber daya mesin *host*. karena dalam proses *deployment*, *docker* akan menjalankan sebuah *container* menggunakan *base image* dengan metode file *system as a layer* yang berarti *docker* hanya akan menyalin lapisan perubahannya saja untuk dijalankan sebagai duplikasi *container* yang berbeda dengan *base image* yang sama.

Dengan diterapkannya sistem virtualisasi *server* berbasis *container*, diharapkan dapat meningkatkan kinerja sebuah *server* dan memudahkan proses *deployment* (penyebaran) aplikasi *web* beserta *software* pendukung seperti *web server*, *database server*, dll ke *server*. Maka permasalahan yang ada diantaranya adalah bagaimana mengimplementasikan *docker* untuk pengelolaan banyak aplikasi *web* di ICT SMK Negeri 1 Rangkasbitung. Jauh daripada itu, akan tercapai efektifitas dan efisiensi kerja mesin karena sumber daya mesin akan digunakan secara maksimal untuk menjalankan layanan yang ada berbasis virtual.

Rumusan masalah dari penelitian ini ialah sebagai berikut:

1. Bagaimana langkah melakukan *virtualisasi docker container* untuk menjamin *web* menjadi *stabil* pada linux ubuntu 18.04 LTS?
2. Bagaimana cara membuat sebuah *server* menjadi lebih efektif dengan *hardware* yang minimum?

p-ISSN : 2406-7733
e-ISSN : 2597-9922

Penelitian ini dilakukan bertujuan untuk:

1. Mengimplementasikan sistem virtualisasi *server* berbasis *docker container*.
2. Melakukan pengujian terhadap efektifitas *docker* dalam penggunaan sumber daya CPU dan memori.
3. Meningkatkan efektifitas penggunaan sumber daya *hardware* mesin *server.*

Penelitian ini dapat memberikan manfaat sebagai berikut:

1. Manfaat Teori pada penelitian ini diharapkan dapat menjadi salah satu ragam referensi penelitian mengenai virtualisasi infrastruktur *server*.
2. Manfaat Praktis pada penelitian ini diharapkan dapat memberikan kemudahan untuk memaksimalkan infrastruktur serta perangkat yang dimiliki oleh SMK Negeri 1 Rangkasbitung.

II. KAJIAN PUSTAKA

M. Fadlulloh RB, dan Asmunin dengan judul Implementasi *Docker* Untuk Pengelolaan Banyak Aplikasi *Web* (2017). Menjelaskan *docker* untuk pengelolaan banyak aplikasi *web*, pada penelitian ini setiap aplikasi *web* beserta *environment* yang dibutuhkan di-*deploy* ke dalam *containers*. Setiap *container* akan memiliki IP *Private* yang hanya dapat diakses oleh *Host*. Sehingga dapat diakses dari luar setiap *container* akan dibuatkan *domain*. Dan untuk mengarahkan *domain* ke *container* tujuan digunakan *Apache* sebagai *reverse proxy.*

Sulastri A, Isnawaty, dan Rizal AS dengan judul Desain dan Implementasi Virtualisasi Berbasis *Docker* untuk *Deployment* Aplikasi *Web* (2018). Menjelaskan bagaimana efektifitas *container Docker* dalam penggunaan sumber daya penyimpanan dan memori dibandingkan dengan mesin virtual tradisional. Selain itu pada penelitian ini akan menguji kompabilitas *container Docker* untuk menjalankan aplikasi yang telah dibangun pada *platform* yang berbeda serta menguji kompabilitas *container Docker* untuk menjalankan aplikasi yang telah dibangun pada lingkungan *host* yang berbeda.

Penelitian ketiga yang dilakukan oleh Firlya A, dan Indri N dengan judul Rancang Bangun Aplikasi *Chatting* Berbasis *Web* Menggunakan *Docker* (2017). Menjelaskan pemanfaatan teknologi docker pada aplikasi *chatting* berbasis *web*. Penelitian ini dimaksudkan untuk dapat memudahkan pengembang aplikasi atau pun *sysadmin* dalam mengembangkan sebuah aplikasi. Sehingga pengembang aplikasi tidak perlu lagi membuat *website* bertuliskan *maintance* karena sedang dalam tahap pengembangan atau perbaikan. Jika menggunakan *docker* pengembang hanya tinggal mengganti sebuah *port* yang ada pada *container* yang ingin di-*up*-kan.

Rakhmi K, Adi M, dan Siti S dengan judul Teknis Kerja *Docker Container* untuk Optimalisasi Penyebaran Aplikasi (2019). Menjelaskan bahwa *docker container* sangat cocok untuk desain arsitektur sistem dengan pendekatan *microservice*, karena masing-masing *service* memiliki lingkungan yang terisolasi namun tetap dapat berkomunikasi satu sama lain. *Microservice* sendiri merupakan suatu pendekatan desain arsitektur sistem informasi yang menspesifikasikan dan memecah fungsi dari sistem yang besar menjadi sistem atau *service* yang kecil dan spesifik. Adapun saran untuk penelitian selanjutnya adalah perlu banyak referensi tentang konfigurasi *web server* desain dan pengembangan aplikasi berbasis *Container* ini karena pada proses pengembangan aplikasi hal ini dapat mempengaruhi desain arsitektur suatu sistem.

Moch. W I S, Rakhmadhany P, dan Widhi dengan judul Implementasi *Load Balancing Server* Basis Data pada Virtualisasi Berbasis Kontainer (2018). Menjelaskan bagaimana merancang sistem *load balancing server* basis data pada virtualisasi berbasis kontainer yang mampu meningkatkan kinerja *server* basis data dalam memenuhi kebutuhan data yang diminta oleh pengguna. Sistem yang dibangun merupakan sebuah sistem *server* basis data yang berjalan di dalam kontainer. Terdapat lima *node server clustering* yang dikelola oleh *kubernetes*. Kemudian dilakukan *deploying* kontainer basis data *PostgreSQL* yang dilakukan menggunakan *image* milik *Cruncydata*. Kontainer basis data *PostgreSQL* dideploy didalam *kubernetes* menggunakan *kind* berupa *statefulset*.

Tanjung P, Dr. Ir. Rendy M, dan Danu DS dengan judul Implementasi dan Analisis *Computer Clustering System* dengan Menggunakan Virtualisasi *Docker* (2017). Menjelaskan bagaimana cara melakukan implementasi layanan *clustering server* dengan menggunakan Docker. Parameter-parameter yang diuji adalah *latency*, *CPU utilization*, dan *throughput*. Pengujian ini dilakukan pada *server* yang memiliki spesifikasi 2 core CPU dan 4 GB RAM, yang pada tulisan ini disebut *server tester*. Pengujian ini dibantu oleh perangkat lunak monitoring *resource server* linux SYSTAT, yang berfungsi sebagai perangkat untuk mengambil data kondisi tiap *node cluster*.

Rivaldy A P, Ratna M, Danu D S dengan judul Implementasi *Web Server Cluster* Menggunakan Metode *Load Balancing* Pada *Container Docker, Lxc,* Dan *Lxd* (2018). Menjelaskan cara melakukan *load balancing* terhadap permintaan *client* berupa HTTP *request* kepada *web server*. Sistem ini menggunakan *HAProxy Load Balancer* sebagai *tools* yang dapat digunakan untuk melakukan proses *load balancing, Apache HTTP server sebagai web server,* dan *container* sebagai wadah untuk pembuat lingkungan virtual serta sistem *clustering* di sisi *server*, adapun *container* yang digunakan yaitu *Docker*, LXC, dan LXD. Sedangkan pada sisi *client* akan dibangkitkan sejumlah *request* kepada *server* menggunakan *software* untuk *load testing* yaitu *Siege*. Pada sistem ini *load balancer* dan *web server* akan dibangun dan dijalankan di atas *container*, yang nantinya akan

p-ISSN : 2406-7733
e-ISSN : 2597-9922

dilakukan perbandingan sejauh mana peningkatan kinerja *server* pada masing-masing *container* yaitu *Docker*, LXC, dan LXD saat dilakukan *load balancing* dengan menggunakan algoritma penjadwalan Round Robin dan penjadwalan *Least Connection* pada *HAProxy Load Balancer*.

Septya A S, Syaifuddin, Diah R dengan judul Pengembangan Mekanisme Akses *E-Learning* Berbasis Linux *Container* (2018). Menjelaskan bagaimana melakukan pengembangan mekanisme akses *E-learning* berbasis *Docker*. Pengaksesan secara langsung oleh *user* terhadap *web server* pada *container* melalui *port* yang dibuka menjadikan beban kerja pada CPU meningkat sehingga pemrosesan *website* menjadi berat jika diakses oleh banyak *user* pada satu waktu. NGINX merupakan *server* HTTP dan *Reverse Proxy* dengan kinerja yang tinggi yang dapat mereduksi penggunaan sumber daya CPU dan memori serta operasi yang stabil. NGINX sebagai *reverse proxy* dapat menjadi solusi *port mapping* pada *Docker* dan juga menyembunyikan alamat internal *server*. Mekanisme kerja NGINX sebagai *reverse proxy* adalah NGINX melakukan perubahan URL yang dikirim oleh *user* menjadi URI yang ditujukan ke *server* dan *web server* akan merespon dengan mengirimkan konten ke NGINX untuk diteruskan ke *user*.

Saefudin dan Septian DN dengan judul Implementasi *Load Balancing Server Web* Berbasis *Docker Swarm* Berdasarkan Penggunaan Sumber Daya *Memory Host* (2019). Menjelaskan bagaimana merancang dan mengimplementasi sebuah sistem *Web Server* dengan mengunakan *Docker Swarm* dengan *fail over* berbadasarkan waktu dan *load balancing* berdasarkan sumber daya komputer. Memori dan CPU merupakan komponen penting dalam *host* dan dapat dijadikan sebagai acuan dalam menentukan kinerja *host* tersebut. Semakin efektif penggunaan memori dan CPU pada *host*, semakin bagus pula kinerjanya. Sebuah *host* dapat dikatakan efektif, jika dapat menggunakan CPU dan memori secukupnya untuk mendapat hasil yang maksimal. Untuk mengetahui pengaruh *loadbalancer* pada *host* terhadap CPU dan memori, dilakukan pengujian dengan menggunakan *load/stress client* dengan beberapa skenario. Analisis pengujian fungsional dilakukan dengan cara menganalisis memonitor pengiriman dan penerimaan data, *output* program dan monitoring *node host* yang *down*. Analisis pengujian fungsional bertujuan untuk mengecek bahwa sistem dapat bekerja sesuai dengan yang diharapkan.

Babak B R, Harrison J B, Mohammad A dengan judul *An Introduction to Docker and Analysis of its Performance* (2017). Menjelaskan analisis performa *docker* dengan melakukan beberapa pengujian dan membandingkannya dengan virtualisasi *server* virtual *machine* berbasis *hypervisor*. Dan hasil yang didapatkan bahwa performa docker lebih baik dibanding dengan *virtual machine* dengan parameter penggunaan *resource* pada mesin *server*.

Virtualisasi Server

Menurut Rakhmi, Adi, dan Siti (2019), Secara sederhana, virtualisasi adalah lawan kata dari *physical machine* atau mesin fisik. *Physical machine* adalah wujud server yang memiliki berbagai komponen seperti *power supply, mainboard, memory, disk*, dan sebagainya. Virtualisasi adalah aplikasi seolah-olah berjalan sendirian dalam satu mesin, tetapi sebenarnya virtualisasi berjalan di atas mesin lain, bersama-sama dengan aplikasi lain.

Menurut Sulastri, Isnawaty, dan Rizal (2018), Virtualisasi adalah sebuah teknik untuk menyembunyikan karakter fisik dari sumber daya komputer dari bagaimana cara sistem lain, aplikasi atau pengguna berinteraksi dengan sumber daya tersebut. Hal ini termasuk membuat sebuah sumber daya tunggal seperti *server*, sebuah sistem operasi, sebuah aplikasi, atau peralatan penyimpanan terlihat berfungsi sebagai beberapa sumber daya logikal, atau dapat juga termasuk definisi untuk membuat beberapa sumber daya fisik (seperti beberapa peralatan penyimpanan atau *server*) terlihat sebagai satu sumber daya logikal. Melihat catatan sejarah istilah virtualisasi mulai dikembangkan sejak pertengahan abad ke 20 dimana hanya digunakan untuk kalangan industri saja.

Virtualisasi / *Virtualization* adalah sebuah teknik atau cara untuk membuat sesuatu dalam bentuk virtualisasi, tidak seperti kenyataan yang ada. Virtualisasi juga digunakan untuk mengemulasikan perangkat fisik komputer, dengan cara membuatnya seolah-olah perangkat tersebut tidak ada (disembunyikan) atau bahkan menciptakan perangkat yang tidak ada menjadi ada.

Terdapat tiga jenis pendekatan virtualisasi untuk membangun *server virtual* yaitu:

1. *Partial Virtualization.* adalah bentuk virtualisasi pada sebagian dari perangkat keras. Perangkat lunak virtualisasi parsial akan mengemulasikan, seolah olah perangkat komputer kita memiliki alat tersebut.
2. *Full Virtualization* berarti membuat seolah-olah ada komputer lain di dalam komputer. Dengan menginstall *Linux* dalam *Windows* Anda, demikian juga menginstall *Windows* dalam *Linux.*
3. *Hardware-assisted Virtualisation.* Merupakan virtualisasi yang didukung oleh *hardware*, jadi ada *hardware* khusus yang berguna untuk meningkatkan *performance* proses virtualisasi. *Hardware-assisted virtualisation* mempunyai *overhead* yang banyak, agar skalabilitas *guest OS* tidak terlalu turun, maka dibantu dengan *hardware.*

Cloud Computing

Istilah *cloud* telah banyak digunakan dalam perkembangan dunia internet karena internet bisa dikatakan sebagai sebuah awan besar. *Cloud computing* sendiri adalah sebuah model komputasi, dimana sumber dayanya seperti *processor / computing*

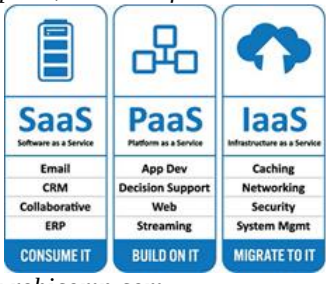
p-ISSN : 2406-7733
e-ISSN : 2597-9922

power, *storage*, *network*, dan *software* dijalankan sebagai layanan melalui media jaringan, bahkan dapat diakses di tempat manapun selama terkoneksi dengan internet.

Cloud computing menerapkan suatu metode komputasi, yaitu kemampuan yang terkait teknologi informasi yang disajikan sebagai suatu layanan yang diakses melalui internet, tanpa mengetahui infrastruktur didalamnya, tenaga ahli yang merancang sistem tersebut atau memiliki kendali atas infrastruktur yang ada (Ernawati, 2013).

Secara umum ada tiga jenis tipe layanan pada *cloud computing*, dimana pada ketiga arsitektur tersebut pengguna tidak mengatur secara langsung yaitu:

1. *Infrastructure as a Service (IaaS).* IaaS menyediakan layanan sampai pada level Sistem Operasi. Jadi pengguna dapat memilih sistem operasi yang akan digunakan dalam bentuk *virtual machine*. Pengguna juga dapat mengatur sumber daya untuk alokasi *hardware* seperti ukuran *memory*, ukuran *hardisk*, dan ukuran *processor*. Contoh dari layanan IaaS adalah *Microsoft Azure IaaS, Amazon EC2, Rackspace Cloud,* dan *Open Stack.*
2. *Platform as a Services (PaaS).* PaaS menyediakan layanan pada level *platform*, jadi pengguna tidak lagi direpotkan dengan instalasi sistem operasi, *web server, database server*, dan aplikasi lainnya. Penyedia layanan PaaS sudah menyediakan sistem operasi lengkap beserta aplikasi yang dibutuhkan untuk *hosting* aplikasi seperti *web server* dan *database server*. Pengguna dapat menggunggah aplikasi yang dibuat melalui panel kontrol yang sudah disediakan. Pengguna juga dapat memilih paket sesuai kebutuhan untuk kebutuhan aplikasi kecil dengan pengguna terbatas, hingga aplikasi dengan pengguna yang besar. Contoh dari layanan PaaS adalah: *Microsoft Azure PaaS* (IIS, ASP.NET, *Open Source technology), Google App Engine, Amazon Elastic Beanstalk, Cloud Foundry,* dan *Heroku.*
3. *Software as a Service (SaaS).* SaaS menyediakan layanan langsung kepada pengguna dalam bentuk aplikasi yang sudah jadi. Bentuk layanan aplikasi yang ditawarkan seperti layanan aplikasi office, email, layanan penyimpanan data, dll. Contoh layanan SaaS adalah: *Office 365, Gmail, Google Docs, DropBox,* dan *Salesforce.*



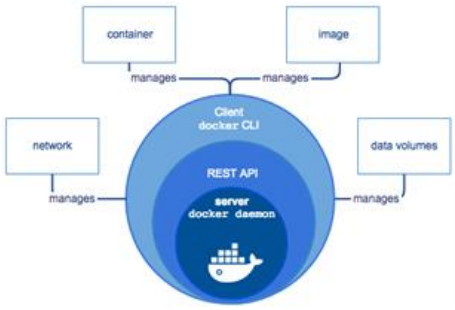
Sumber: www.robicomp.com

Gambar 1. Layanan *Cloud Computing*

Docker

Docker adalah sebuah *project open-source* yang menyediakan *platform* terbuka untuk *developer* maupun *sysadmin* untuk dapat membangun, mengemas, dan menjalankan aplikasi dimanapun di dalam sebuah *container*.

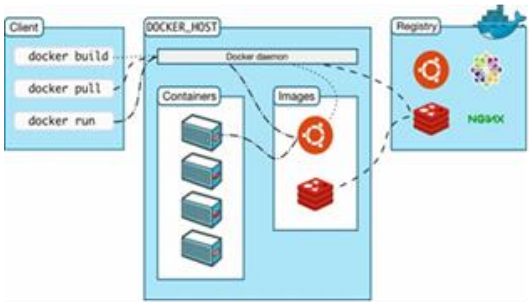
Docker menggunakan arsitektur *client-server*. Dimana *client* dan *docker* berkomunikasi dengan *daemon docker*, yang melakukan suatu tindakan untuk membangun, menjalankan, dan mendistribusikan *container docker*. *Client docker* dan *daemon* dapat berjalan pada sistem yang sama. *Client docker* dan *daemon* berkomunikasi menggunakan REST API, melalui soket UNIX atau antarmuka jaringan.



Sumber: www.docs.docker.com

Gambar 2. Teknis Kerja *Docker*

Arsitektur *docker* mempunyai beberapa komponen yaitu terdiri dari *docker daemon, docker client, docker images, docker container* dan *docker registry*. *Docker* menggunakan teknologi *client-server* untuk menghubungkan antara *docker client* dan *docker daemon.*



Sumber: www.docs.docker.com

Gambar 3. Arsitektur *Docker*

Docker sendiri memiliki banyak komponen dalam menjalankan proses dan tugasnya. Masing-masing komponen pada *docker* itu saling ketergantungan, sehingga jika ada komponen yang tidak tersedia hal itu tentu saja akan mempengaruhi kinerja *docker*.

Berikut merupakan komponen yang terdapat pada docker:

1. *Docker Daemon* adalah sebuah *service* yang dijalankan di dalam *host* dalam *Operating System* (OS). Fungsinya adalah membangun, mendistribusikan, dan menjalankan *container docker*. Pengguna tidak dapat langsung menggunakan *docker daemon*, akan tetapi untuk menggunakan *docker daemon* maka pengguna

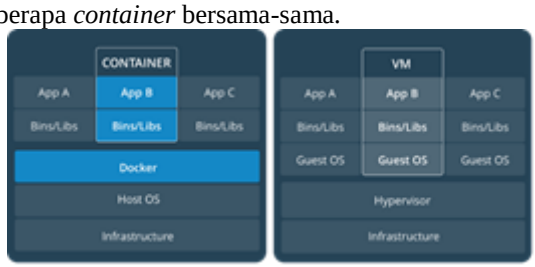
p-ISSN : 2406-7733
e-ISSN : 2597-9922

menggunakan *docker client* sebagai perantara atau CLI.

2. *Docker Client* adalah seperangkat perintah *command line* untuk mengoperasikan *docker container*, misalnya membuat *container*, *start/stop container*, menghapus (*destroy*), dan sebagainya yang ada di *docker daemon*.
3. *Docker Images* adalah sebuah komponen docker yang berupa *template* dasar yang bersifat *read only*. *Template* ini sebenarnya adalah sebuah OS atau OS yang telah di-*install* berbagai aplikasi. *Docker images* berfungsi untuk membuat *docker container*, dengan hanya satu *docker images* dapat membuat banyak *docker container*.
4. *Docker Container* merupakan sebuah *images* yang dapat dikemas dan bersifat *read-write*, *container* berjalan di atas *images*. *Docker container* juga bisa dikatakan sebagai sebuah *folder*, dimana *docker container* ini dibuat dengan menggunakan *docker container*. Setiap *docker container* disimpan maka akan terbentuk *layer* baru tepat di atas *docker images* atau *base images* di atasnya.
5. *Docker Registry* merupakan *repository* distribusi kumpulan *docker image*s yang bersifat *private* maupun *public* yang dapat di akses melalui *docker hub*.

Container adalah virtualisasi pada level sistem operasi dimana tiap proses atau aplikasi yang dijalankan tiap *container* memiliki kernel yang sama. Hal ini menjadi keuntungan sendiri dibandingkan virtualisasi pada level mesin (*virtual machine*), dimana *virtual machine* membutuhkan kernel sistem operasi yang berbeda-beda tiap aplikasi yang dijalankan.

Perbedaan *docker* dengan *virtual machine* adalah *container* dalam *docker* memiliki sumber daya yang terisolasi sama dan manfaat alokasi seperti *virtual machine* tetapi perbedaan arsitektur yang berbeda memungkinkan *containers* untuk menjadi lebih portabel dan efisien. *virtual machine* membutuhkan banyak ruang untuk *guest OS* atau *operating system*. Sedangkan *docker* tidak membutuhkan banyak ruang untuk *operating system*. Karena *operating system* pada *docker* dapat digunakan untuk menjalankan beberapa *container* bersama-sama.



Sumber: www.docs.docker.com

Gambar 4. Perbandingan *Docker* dan *Virtual Machine*

Docker mempunyai beberapa keuntungan, beberapa keuntungan menggunakan *docker* yaitu sebagai berikut:

1. Kemudahan dalam Konfigurasi. Dengan adanya *docker*, dapat menjalankan sistem virtualisasi dengan sangat mudah tanpa melakukan banyak konfigurasi. Hanya cukup menggunakan satu baris perintah maka mesin *container* pun sudah dapat berjalan dengan lancar.
2. Isolasi. Sistem aplikasi *docker* yang terisolasi membuat *container* menjadi aman. Proses kerja *container docker* yang tidak akan mengganggu *container* yang lain juga merupakan keuntungan yang dapat dimanfaatkan terutama oleh para *developer*.
3. Standarisasi Lingkungan dan Kontrol Versi. Adanya kemudahan dalam melakukan pengembangan dan rilis dari sistem. *Docker* yang menggunakan *repository* seperti GIT tentu akan memudahkan dalam hal control versi dan pengembangan.
4. *Multi Cloud Platform*. Salah satu manfaat terbesar docker adalah portabilitas. Selama beberapa tahun terakhir, semua penyedia *cloud computing* terbesar, termasuk *Amazon Web Services* (AWS) dan *Google Compute Platform* (GCP), telah merangkul *docker* dan menambahkan dukungan individu.

Prometheus

Prometheus adalah sebuah *time-series database open source* yang biasa digunakan untuk memonitoring sistem dan digunakan sebagai *alerting toolkit* atau alat peringatan pada sebuah sistem. *Prometheus* telah digunakan sejak tahun 2012 dan sudah cukup banyak perusahaan atau organisasi yang mengadopsi *Prometheus* untuk sistemnya. *Prometheus* sekarang merupakan *open source project* yang sudah berdiri sendiri. *Promotheus* berbasis *Pull System*, dimana *server Promotheus* yang akan "meminta" data dari aplikasi yang sedang berjalan secara berkala. *Prometheus* mempunyai model data berupa *key-value* sebagai label, yang disebut *tags*. Dan *Prometheus support timestamps* sampai ke resolusi *milisecond* serta hanya *support* tipe data float64 (Bastos, 2019).

Grafana

Grafana adalah perangkat analisis dan visualisasi metrik berbasis *open source*. *Grafana* paling sering digunakan untuk memvisualisasikan data deret waktu untuk infrastruktur dan analisis aplikasi. Namun Grafana tidak hanya sebatas hal itu saja, kerap kali servis digunakan untuk visualisasi sensor industri, pengimplementasian *Internet of thing* (IoT), pengamatan cuaca dan pengontrolan proses yang sedang berjalan. Ada berbagai macam opsi penataan dan pemformatan yang diekspos setiap panel untuk memungkinkan pembuatan desain yang sempurna. Terdapat lima jenis panel dalam *Grafana*, yaitu grafik, *singlestat*, *dashlist*, tabel dan teks. Panel grafik memungkinkan untuk membuat grafik metric dan seri sebanyak yang diinginkan. Panel *singlestat* membutuhkan pereduksian sebuah *query* ke sebuah nomor. *Dashlist* dan teks adalah panel khusus yang

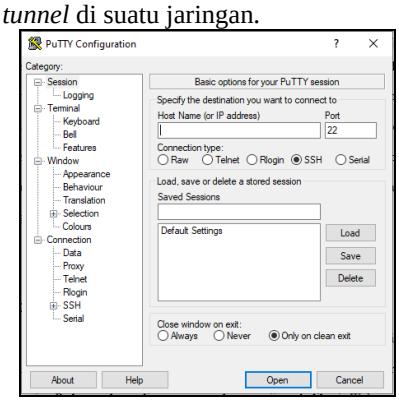
p-ISSN : 2406-7733
e-ISSN : 2597-9922

tidak terhubung ke sumber data apapun (Karolus, 2018).

Panel dapat dibuat lebih dinamis dengan memanfaatkan *string variable Dashboard* templating dalam konfigurasi panel. Grafana mendukung banyak *backend* penyimpanan yang berbeda untuk setiap data time series (*Data Source*). Setiap sumber data memiliki editor *Query* khusus yang disesuaikan untuk fitur dan kemampuan dari data *source* tertentu. Setiap *Query* dari masing-masing *data source* tentu berbeda, namun di dalam *Grafana* dapat disatukan dalam satu *Dashboard* dengan catatan masing-masing panel terikat pada satu data *source*. Beberapa data *source* telah secara resmi didukung oleh *service* dari Grafana, antara lain: *Graphite, InfluxDB, OpenTSDB, Prometheus, Elasticsearch*, dan *CloudWatch. Grafana* hadir dengan *plugin data source* yang sangat lengkap untuk *InfluxDb*. *Grafana* mendukung berbagai macam *editor query* dengan fitur yang kaya, anotasi dan *templating queries* (Teguh Indra, 2018).

PuTTY

PuTTY adalah sebuah program *open source* yang dapat digunakan untuk melakukan protokol jaringan SSH, *Telnet*, dan *Rlogin*. Protokol ini dapat digunakan untuk menjalankan sesi *remote* pada sebuah komputer melalui sebuah jaringan, baik itu LAN, maupun *internet*. Program ini banyak digunakan oleh komputer tingkat menengah ke atas, yang biasanya digunakan untuk menyambungkan, mensimulasi, atau mencoba berbagai hal yang terkait dengan jaringan. Program ini juga dapat digunakan sebagai *tunnel* di suatu jaringan.



Sumber: Dokumentasi Pribadi

Gambar 5. Tampilan Awal Putty

Apache Jmeter

Aplikasi Apache JMeter adalah perangkat lunak *open source*, aplikasi Java murni 100% dirancang untuk memuat perilaku fungsional tes dan mengukur kinerja. Pada awalnya dirancang untuk pengujian Aplikasi *Web* tetapi sejak diperluas untuk menguji fungsi lainnya. *Apache JMeter* dapat digunakan untuk menguji kinerja baik pada sumber daya statis dan dinamis *(Web services (SOAP / REST), Web* bahasa dinamis - *PHP, Java, ASP.NET, File,* dll. Hal ini dapat digunakan untuk mensimulasikan beban berat pada *server*, sekelompok *server*, jaringan atau objek untuk menguji kekuatan atau untuk menganalisis kinerja secara keseluruhan di bawah jenis beban yang berbeda. Dapat menggunakannya untuk membuat analisis grafis kinerja atau untuk menguji perilaku / objek *server* / *script* di bawah beban bersamaan berat.

Linux OS

Linux adalah nama yang diberikan kepada sistem operasi komputer bertipe *Unix. Linux* merupakan salah satu contoh hasil pengembangan perangkat lunak bebas dan sumber terbuka utama. Seperti perangkat lunak bebas dan sumber terbuka lainnya pada umumnya, kode sumber *Linux* dapat dimodifikasi, digunakan dan didistribusikan kembali secara bebas oleh siapa saja. Nama "*Linux*" berasal dari nama pembuatnya, yang diperkenalkan tahun 1991 oleh Linus Torvalds. Sistemnya, peralatan sistem dan pustakanya umumnya berasal dari sistem operasi GNU, yang diumumkan tahun 1983 oleh Richard Stallman. Kontribusi GNU adalah dasar dari munculnya nama *alternative GNU/Linux*.

Linux adalah sistem operasi berbasis *GNU/Linux* yang bersifat *Open Source* dan memiliki banyak varian seperti *Debian, Slackware, Open Suse, Archlinux, Redhat* dan sebagainya. Walaupun sangat banyak varian *GNU/Linux* hanya menyediakan aplikasi yang sudah ditentukan yang mungkin kurang bermanfaat oleh pengguna sehingga hal ini mengakibatkan banyak pengguna yang melakukan *remastering* untuk memenuhi kebutuhannya. *Remastering* adalah proses membuat sistem operasi baru dengan mengurangi atau menambahkan fitur-fiturnya dari distro *GNU/Linux* yang telah ada.

Web Server

Web Server adalah sebuah perangkat lunak *server* yang berfungsi menerima permintaan HTTP atau HTTPS dari klien yang dikenal dengan *web browser* dan mengirimkan kembali hasilnya dalam halaman-halaman *web* yang umumnya berbentuk dokumen HTML . *Web server* yang dimaksud disini adalah simulasi dari sebuah web *server* secara fisik. *Web server* biasanya juga disebut HTTP *server* karena menggunakan protocol HTTP sebagai basisnya (Kurniawan, 2008).

Web server adalah *software* yang memberikan layanan berbasis data yang berfungsi menerima *request* atau permintaan atau HTTP *request* dari *client* dalam bentuk HTTP yang berasal dari *web browser* dan akan mengirimkan data yang diminta atau HTTP *respond* dalam bentuk halaman *web* beserta konten-konten yang diinginkan dalam bentuk dokumen HTML. Fungsi utama dari *web server* sendiri adalah melakukan transfer permintaan atau *request client* melalui aturan atau protokol yang telah ditentukan lalu mengirimkan data yang diminta *client* kembali dalam bentuk halaman *web* serta konten yang di-*request*.

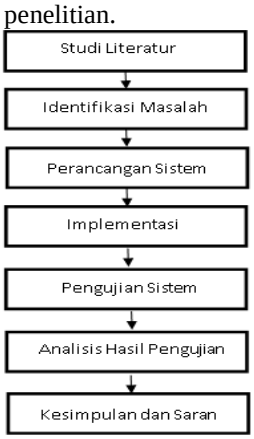
Berikut ini merupakan beberapa macam *web server*, diantaranya adalah:

p-ISSN : 2406-7733
e-ISSN : 2597-9922

1. *Apache* adalah *web server* yang berbasis *open source* dan banyak digunakan karena memiliki pengguna yang sudah banyak sehingga juga memiliki dokumentasi yang lebih lengkap dibandingkan *web server* lainnya. Ciri spesifik dari *Apache* adalah *web server* ini menggunakan arsitektur *keep-alive* atau *forked usered* dalam melayani suatu *request*, sehingga lebih memakan konsumsi sumber daya. *Apache* memiliki beberapa fitur seperti kontrol akses, CGI, PHP, SSI, URL Rewritting dan lainnya. *Apache* dapat berjalan baik secara multi prosess maupun multi thread arsitektur dimana dapat menjalankan proses yang berjalan secara konkkuren. Dalam memproses halaman dinamik, *Apache* merupakan *web server* yang memiliki performa terbaik dibandingkan dengan *web server* lain (Apache, 2018).
2. *Nginx* adalah *software open-sourc*e yang memiliki kinerja tinggi sebagai *server* HTTP dan *reverse proxy*. Nginx dengan cepat memberikan konten statis dengan penggunaan efisien sumber daya sistem. Hal ini dapat menyebarkan dinamis HTTP konten di jaringan.
3. IIS (*Internet Information Services*) digunakan dalam OS *Windows* (Windows 2000 dan *Windows* 2003) yang didukung dengan komponen seperti protokol jaringan DNS, TCIP/IP dan *software* yang digunakan untuk membuat situs *web*. IIS juga kompatibel untuk mendukung beberapa protokol seperti FTP, HTTP, NNTP, SMTP dan SSL. Kelebihan dari *server* ini bisa diakses pada penuh pada *Windows* dan mendukung *platform* .NET. Kekurangannya adalah *web server* ini termasuk jenis yang berbayar.

III. METODE PENELITIAN

Tipe penelitian yang dipilih yaitu penelitian terapan, yang bertujuan untuk memberikan solusi serta pengembangan dalam infrastruktur *server* yang ada di ICT SMK Negeri 1 Rangkasbitung. Pada gambar 6 merupakan alur tahapan penelitian yang digunakan dalam penelitian.



Sumber: Dokumentasi Pribadi

Gambar 6. Skema Tahapan Penelitian

Pada gambar 6 menjelaskan alur tahapan penelitian yang digunakan dalam penelitian. Penjelasan dari setiap langkahnya adalah sebagai berikut:

1. Survei Literatur. Tahap ini adalah melakukan pengumpulan bahan literatur dan informasi berkaitan dengan judul penelitian. Mempelajari literatur dari berbadai bidang ilmu yang sesuai dengan penelitian. Literatur tersebut didapat dari buku, jurnal, penelitian sebelumnya dan dokumentasi yang ditemukan selama penelitian.
2. Identifikasi Masalah. Masalah yang diteliti dalam penelitian ini adalah bagaimana merancang sistem virtualisai *server* berbasis *docker container* yang mampu meningkatkan efektifitas kinerja *server*.
3. Perancangan Sistem. Menganalisis kebutuhan sistem untuk mempersiapkan dalam melakukan implementasi dari sebuah sistem yang sedang diteliti. Yaitu sistem virtualisasi *server* berbasis *docker container*.
4. Implementasi. Tahap implementasi merupakan tahap selanjutnya dari tahapan perancangan. Pada tahap ini sistem dibangun berdasarkan rancangan yang telah dibuat untuk membangun virtualisai *server* berbasis *docker container*.
5. Pengujian Sistem. Pengujian sistem dilakukan setelah implementasi telah dilaksanakan. Tujuan dari uji coba sistem ini adalah memastikan hasil dari implementasi sistem yang telah dibuat sesuai dengan perancangan yang telah dilakukan. Apabila terjadi kesalahan yang menyebabkan kegagalan dalam uji coba maka akan dilakukan kembali perancangan serta mengulang implementasi.
6. Analisis Hasil Pengujian. Tahap analisis dilakukan setelah pengujian sistem dilakukan. Pada tahap ini merupakan tahap analisid hasil data-data yang didapatkan dari tahap pengujian.
7. Kesimpulan dan Saran. Kesimpulan ini dari hasil pengujian dan analisis metode. Tahap akhir dari penulisan ini adalah saran yang dimaksudkan untuk memperbaiki kesalahan yang terjadi serta memberikan pertimbangan untuk pengembangan selanjutnya.

IV HASIL DAN PEMBAHASAN

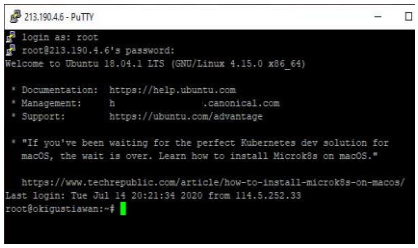
Hasil Penelitian

Untuk mendapatkan hasil penelitian sesuai yang dibutuhkan, dilakukan beberapa tahapan pengujian baik itu melalui *host docker* ataupun komputer *client* yang terkoneksi dengan *host docker*. Berikut adalah langkah dan hasil pengujian *docker*.

Hasil Pengujian Fungsional

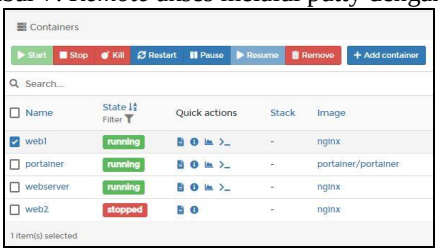
Hasil dari pengujian fungsional yang dilakukan dengan interaksi antara *client-server* dan *server-client* adalah sebagai berikut:

p-ISSN : 2406-7733
e-ISSN : 2597-9922



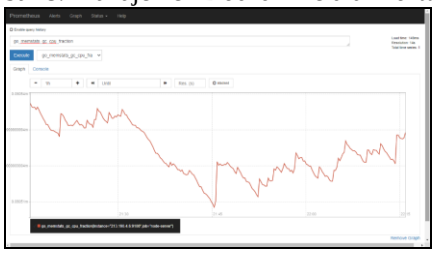
Sumber: Dokumentasi Pribadi

Gambar 7. *Remote* akses melalui putty dengan ssh



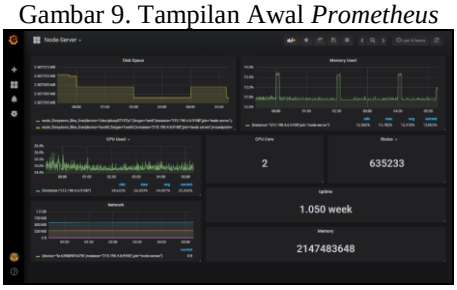
Sumber: Dokumentasi Pribadi

Gambar 8. Manajemen *Docker* Melalui *Portainer*



Sumber: Dokumentasi Pribadi

Gambar 9. Tampilan Awal *Prometheus*



Sumber: Dokumentasi Pribadi

Gambar 10. Monitoring *Server* Melalui *Grafana*

Untuk rincian hasil pengujian fungsional docker yang dilakukan dengan interaksi antara *client-server* pada penelitian ini dapat dilihat pada tabel 1.

Tabel 1 Hasil Pengujian Fungsional

No	Kebutuhan Fungsional	Hasil
1.	Pengguna melakukan *remote* akses ke *server* dengan *login* menggunakan SSH	OK
2.	Pengguna menambahkan *image* baru dari sebuah aplikasi ke *server* dari *docker registry*	OK
3.	Pengguna membuat *container* baru dari sebuah image yang telah ditambahkan dari *docker registry*	OK
4.	Pengguna menjalankan *container* yang telah dibuat	OK
5.	Pengguna menghentikan *container* yang telah dibuat	OK
6.	Pengguna menghapus *container* yang telah dibuat	OK
7.	Pengguna bisa menjalankan *container* melalui fitur yang ada pada *dashboard portainer*	OK
8.	Pengguna bisa menghentikan *container* melalui fitur yang ada pada *dashboard portainer*	OK
9.	Pengguna bisa menghapus *container* atau *image* melalui fitur yang ada pada *dashboard portainer*	OK
10.	Akses monitoring *server* menggunakan *grafana*	OK

Sumber: Dokumentasi Pribadi

Hasil Pengujian Penggunaan CPU

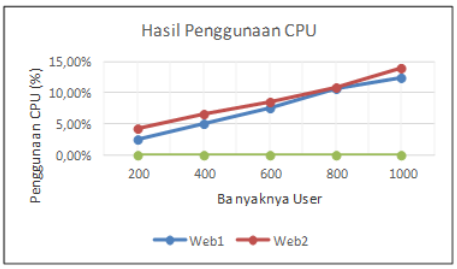
Penggunaan CPU yang diukur adalah penggunaan CPU paling tinggi pada setiap pengujian menggunakan variasi banyaknya jumlah *user* yang berbeda pada masing-masing *container* yang berjalan. Data yang diambil berasal dari statistik *web* yang sedang berjalan ketika proses pengujian berlangsung.

Untuk rincian hasil pengujian penggunaan CPU yang dilakukan dengan interaksi dengan menggunakan perangkat lunak *apache jmeter* pada penelitian ini dapat dilihat pada tabel 2 dan gambar 11.

Tabel 2. Hasil Pengujian Penggunaan CPU (%)

User Request	Nama *Container*	
	Web 1	Web 2
200	2.54 %	4.19 %
400	4.99 %	6.49 %
600	7.54 %	8.58 %
800	10.62 %	10.84 %
1000	12.35 %	13.89 %

Sumber: Dokumentasi Pribadi



Sumber: Dokumentasi Pribadi

Gambar 11. Grafik Hasil Pengujian Penggunaan CPU

Hasil Pengujian Penggunaan Memori

Penggunaan memori yang diukur adalah penggunaan memori paling tinggi pada setiap pengujian menggunakan variasi banyaknya jumlah *user* yang berbeda pada masing-masing *container* yang berjalan. Data yang diambil berasal dari statistik web yang sedang berjalan ketika proses pengujian berlangsung.

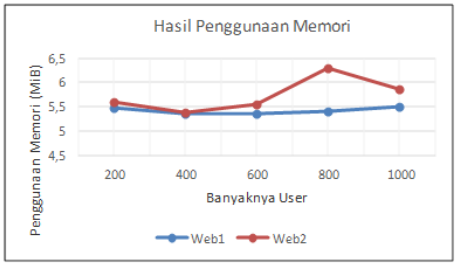
Untuk rincian hasil pengujian penggunaan memori yang dilakukan dengan interaksi dengan menggunakan perangkat lunak *apache jmeter* pada penelitian ini dapat dilihat pada tabel 3 dan gambar 12.

Tabel 3. Hasil Pengujian Penggunaan Memori (MiB)

User Request	Nama *Container*	
	Web 1	Web 2
200	5.492 MiB	5.598 MiB
400	5.367 MiB	5.383 MiB
600	5.367 MiB	5.559 MiB
800	5.398 MiB	6.297 MiB
1000	5.508 MiB	5.875 MiB

Sumber: Dokumentasi Pribadi

p-ISSN : 2406-7733
e-ISSN : 2597-9922



Sumber: Dokumentasi Pribadi

Gambar 12. Grafik Hasil Pengujian Penggunaan Memori

Pembahasan Penelitian

Berdasarkan pengujian yang telah dilakukan dengan cara melakukan variasi banyaknya jumlah *user request* yang berbeda pada masing-masing *container*, yaitu sebanyak 200, 400, 600, 800, dan 1000 pengguna. Hasil dari penggunaan CPU berada dibawah 15% dan penggunaan memori berada dibawah 7 MiB. Proes pengujian ini dapat dilihat pada gambar 4.12 dengan metode pengujian variasi jumlah *user request* menggunakan perangkat lunak *apache jmeter.* Hal ini dapat membuktikan bahwa kemampuan *docker* dalam pemanfaatan sumber daya *hardware* sangat baik dan lebih efisien sehingga dapat mengoptimalisasikan infrastruktur yang ada.

V. PENUTUP

Kesimpulan

Berdasarkan penelitian yang telah dilakukan, dapat diambil beberapa kesimpulan, yaitu:

1. Dengan adanya penerapan virtualisasi *server* berbasis *docker container* pada ubuntu 18.04 LTS, dapat menjamin *web* menjadi stabil. Karena pada docker container memastikan aplikasi dan sumber daya yang terisolasi serta terpisah sehingga para penggunanya dapat menyesuaikan kebutuhan di setiap aplikasi tanpa perlu mempengaruhi konfigurasi pada aplikasi yang lain.
2. Adanya pemanfaatan *docker container* pada perancangan *server* dapat memanfaatkan *hardware* yang ada untuk digunakan secara maksimal. Karena pada *docker container* dimana kernel yang digunakan adalah dari bagian sistem operasi *host* nya sendiri, sehingga tidak membebani kinerja dari *server host.*

Saran

Menyadari bahwa di dalam penelitian ini banyak sekali kelemahan dan kekurangan, oleh sebab itu penulis ingin menyarankan beberapa hal untuk memperbaiki kekurangan dan kelemahan yaitu

1. Mengamankan komunikasi antar *server* karena saat ini *ip public* server bisa diakses oleh siapapun. Hal tersebut bisa dilakukan dengan mengimplementasikan *private* IP untuk komunikasinya.
2. Mengembangkan kembali konfigurasi-konfigurasi *docker* seperti *dockerfile, docker commit* dan *docker-compose* dengan *service* yang nantinya dibutuhkan saat pengembangan.

DAFTAR PUSTAKA

Adinta, F., & Neforawati, I. (2019). "Rancang Bangun Aplikasi Chatting Berbasis Web Menggunakan Docker". *JOISIE (Journal Of Information Systems And Informatics Engineering), Vol. 1* (No.1), 28-34

Alauddin, M. F. (2017). "Implementasi Virtual Data Center Menggungakan Linux Container Berbasis Docker dan SDN". *Doctoral dissertation, Institut Teknologi Sepuluh Nopember, Vol. 6* (No.2), A440-A442.

Al Fatta, H., & Marco, R. (2015). "Analisis pengembangan dan perancangan sistem informasi akademik smart berbasis cloud computing pada sekolah menengah umum negeri (smun) di daerah istimewa yogyakarta". Telematika. *Jurnal Telematika, Vol. 8* (No.2), 63-91.

Apridayanti, S., Isnawaty, I., & Saputra, R. A. (2018). "Desain Dan Implementasi Virtualisasi Berbasis Docker Untuk Deployment Aplikasi Web". *semanTIK, Vol. 4* (No.2), 37-46.

Asokan, M., & Arul, P. (2015). "Load Testing For Jquery Based Mobile Websites Using Borland Silk Performer". *International Journal of Computer Engineering & Technology (IJCET), Vol. 6* (No.9), 12-20.

Aziz, A., & Tampati, T. (2015). "Analisis Web Server untuk Pengembangan Hosting Server Institusi: Pembandingan Kinerja Web Server Apache dengan Nginx". *Analisis Web Server untuk Pengembangan Hosting Server Institusi: Pembandingan Kinerja Web Server Apache dengan Nginx, Vol. 1* (No.2) 12-20.

Bella, M. R. M., Data, M., & Yahya, W. (2019). "Implementasi Load Balancing Server Web Berbasis Docker Swarm Berdasarkan Penggunaan Sumber Daya Memory Host". *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, Vol. 3* (No.4), 3478-3487.

Bik, R., & Fadlulloh, M. (2017). "Implementasi Docker Untuk Pengelolaan Banyak Aplikasi Web (Studi Kasus: Jurusan Teknik Informatika Unesa)". *Jurnal Manajemen Informatika, Vol. 7* (No.2), 46-50.

Chandra, A. Y. (2019). "Analisis Performansi Antara Apache & Nginx Web Server dalam Menangani Client Request". *Jurnal Sistem dan Informatika (JSI), Vol. 14* (No.1), 48-56.

Erinton, R., Negara, R. M., & Sanjoyo, D. D. (2017). "Analisis Performasi Framework Codeigniter Dan Laravel Menggunakan Web Server Apache". *eProceedings of Engineering, Vol. 4* (No.3), 3565-3572.

p-ISSN : 2406-7733
e-ISSN : 2597-9922

Goasguen, Sebastien. (2016). *Docker Cookbook*. United States of America: O'Reilly Media.

Hakim, D. K., Yulianto, D. Y., & Fauzan, A. (2019). "Pengujian Algoritma Load Balancing pada Web Server Menggunakan Nginx". *JRST (Jurnal Riset Sains dan Teknologi), Vol. 3* (No.2), 85-92.

Halawa, S. (2016). "Perancangan Aplikasi Pembelajaran Topologi Jaringan Komputer untuk Sekolah Menengah Kejuruan (SMK) Teknik Komputer dan Jaringan (TKJ) dengan Metode Computer Based Instruction". *JURIKOM (Jurnal Riset Komputer), Vol. 3* (No.1), 66-71.

Harjono, E. B. (2017). "Analisa Dan Implementasi Dalam Membangun Sistem Operasi Linux Menggunakan Metode LSF Dan REMASTER". *Jurnal & Penelitian Teknik Informatika, Vol. 1* (No.1), 30-35.

Jusuf, H. (2015). "Penggunaan Secure Shell (SSH) Sebagai Sistem Komunikasi Aman Pada Web Ujian Online". *Bina Insani ICT Journal, Vol. 2* (No.2), 75-84.

K, Arun. B, Vinutha. & B, Vinayaditya. (2019). "Real Time Monitoring Of Servers With Prometheus And Grafana For High Availability". *International Research Journal of Engineering and Technology (IRJET), Vol. 6* (No.4) 5093-5096.

Kamarudin, K., Kusrini, K., & Sunyoto, A. (2018)." Uji Kinerja Sistem Web Service Pembayaran Mahasiswa Menggunakan Apache JMeter (Studi Kasus: Universitas AMIKOM Yogyakarta)". *Jurnal Teknologi Informasi, Vol. 8* (No.1) 44-52.

Khadafi, S., Meilani, B. D., & Hidayat, S. A. (2017). "Pengukuran Kompatibilitas Performa Komputer Server Menggunakan Jmeter Pada Raspberry Pi Dan Pc Sebagai Layanan Web Server". *In Prosiding Seminar Nasional Sains dan Teknologi Terapan, Vol.* (No.) C157-C162.

Khairil, K., Riyanto, N. P., & Rosmeri, R. (2013). "Membangun Webserver Intranet Dengan Linux (Studi Kasus Di Laboratorium Komputer SMP Negeri 38 Seluma Bengkulu Selatan)". *Media Infotama, Vol. 9* (No.1), 1-21.

Khalida, R., Muhajirin, A., & Setiawati, S. (2019). "Teknis Kerja Docker Container untuk Optimalisasi Penyebaran Aplikasi". *PIKSEL: Penelitian Ilmu Komputer Sistem Embedded and Logic, Vol. 7* (No.2), 167-176

Kurniawan, M Ilham. (2020). *Virtualisasi Dengan Docker*. Jakarta: Institut STMIK Bina Sarana Global.

Kusuma, T. P., Munadi, R., & Sanjoyo, D. D. (2017)." Implementasi Dan Analisis Computer Clustering System Dengan Menggunakan Virtualisasi Docker". *eProceedings of Engineering, Vol. 4* (No.3), 3548-35

Permatasari, D. I., Ardani, M., Ma'ulfa, A. Y., Ilhami, N., dkk. (2020). "Pengujian Aplikasi Menggunakan Metode Load Testing dengan Apache Jmeter pada Sistem Informasi Pertanian". *JUSTIN (Jurnal Sistem dan Teknologi Informasi), Vol. 8* (No.1) 135-139.

Permatasari, D. I., Santoso, B., Ningtias, N., YR, M. H., Atika, R., Widad, N., & Maulana, I. (2019). "Pengukuran throughput load testing menggunakan test case sampling gorilla testing". *In Seminar Nasional Sistem Informasi (SENASIF), Vol. 3* (No.1) 2008-2014.

Pratama, R. A., Mayasari, R., & Sanjoyo, D. D. (2018). "Implementasi Web Server Cluster Menggunakan Metode Load Balancing Pada Container Docker, Lxc, Dan Lxd". *eProceedings of Engineering, Vol. 5* (No.3), 5028-5035.

Putra, A. D. Widhi, Y., & Bhawiyuga, A. (2018). "Analisis Kinerja Dan Konsumsi Sumber Daya Aplikasi Web Server Pada Platform Raspberry Pi". *Doctoral dissertation, Universitas Brawijaya, Vol. 3* (No.4) 3512-3521.

Rad, B. B., Bhatti, H. J., & Ahmadi, M. (2017). "An introduction to docker and analysis of its performance". *International Journal of Computer Science and Network Security (IJCSNS), Vol. 17* (No.3), 228-235.

Razi, M. F. (2017). "Implementasi Pengendali Elastisitas Sumber Daya Berbasis Docker Untuk Aplikasi Web". Tesis Institut Teknologi Sepuluh Nopember. Tidak Diterbitkan.

Sakti, B., Aziz, A., & Doewes, A. (2013). "Uji Kelayakan Implementasi SSH sebagai Pengaman FTP Server dengan Penetration Testing". *ITSMART: Jurnal Teknologi dan Informasi, Vol. 2* (No.1), 44-51.

Santosa, M. W. I., Primananda, R., & Yahya, W (2018). "Implementasi Load Balancing Server Basis Data Pada Virtualisasi Berbasis Kontainer". *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, Vol. 2* (No.12), 6908-6914.

Sardi, E. S. M. (2017). "Implementasi Teknik Virtualisasi Container Dengan Docker Untuk Pengelolaan Aplikasi Web Di Dinas Komunikasi Dan Informatika Kota Payakumbuh". Tugas Akhir Politeknik Negeri Padang. Tidak Diterbitkan.

Satwika, I., & Semadi, K. N. (2020). "Perbandingan Performansi Web Server Apache Dan Nginx Dengan Menggunakan Ipv6". *SCAN-Jurnal Teknologi Informasi dan Komunikasi, Vol. 15* (No.1), 10-15.

Sugianto, Masim Vavai., Marsan susanto dkk (2016) : Virtualisasi Modern Berbasis Docker. Bekasi Timur : PT. Excellent Infotama Kreasindo.

Sumbogo, Y. T., Data, M., & Siregar, R. A. (2018). "Implementasi Failover Dan Autoscaling Kontainer Web Server Nginx Pada Docker Menggunakan Kubernetes". *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, Vol. 2* (No.12), 6849-6854.

Suryanto, S. A. (2018). "Pengembangan Mekanisme Akses E-Learning Berbasis Linux Container". *JIPI*

p-ISSN : 2406-7733
e-ISSN : 2597-9922

(Jurnal Ilmiah Penelitian dan Pembelajaran Informatika), Vol. 3 (No.1), 45-52.

Utomo, Eko Priyo. (2006). *Pengantar Jaringan Komputer Bagi Pemula*. Bandung: Yrama Widya.

.