SKANIKA: Sistem Komputer dan Teknik Informatika
Volume 7, Nomor 2, Juli 2024, Halaman 224-235 E-ISSN: 2721-4788

Penerapan *Horizontal Pod Autoscaler* dan *Redis Cluster* Berbasis Kubernetes untuk Meningkatkan Performa Website Elearning

Diki Taufik Gurohman[1], Bekti Maryuni Susanto[2*], Agus Hariyanto[3], Ery Setiyawan Jullev Atmadji[4], Mukhamad Angga Gumilang[5], Ely Antika[6], Nanik Anita Mukhlisoh[7]
[1,2,3,4,5,6,7] Teknologi Informasi, Politeknik Negeri Jember, Jember, Indonesia
E-mail: e32190192@student.polije.ac.id, [2*]bekti@polije.ac.id, [3]agus_hariyanto@polije.ac.id, [4]ery@polije.ac.id, [5]angga.gumilang@polije.ac.id, [6]elly_antika@polije.ac.id, [7]nanik_anita@polije.ac.id
(* : corresponding author)

Abstrak

Elearning merupakan sarana yang sangat vital dalam pembelajaran saat ini. Untuk memberikan layanan yang optimal, server elearning membutuhkan sumber daya komputasi yang cukup besar ketika *user* mengakses secara bersamaan dalam jumlah besar. Namun, mahalnya sumber daya komputasi seperti CPU, *memory* dan *disk* penyimpanan membuat organisasi kesulitan dalam memenuhi kebutuhan pengguna yang besar. Penelitian sebelumnya membandingkan performa dua *public cloud* pada *learning management system* berbasis moodle. Hasilnya waktu *backup* dan *restore* akan meningkat sekitar 10 detik untuk setiap ukuran data tambahan sebesar 500 MB. Penelitian ini bertujuan menerapkan horizontal *pod autoscaler* dan *redis cluster* berbasis Kubernetes pada server *elearning* Moodle. Moodle digunakan untuk menjalankan *elearning* dan redis sebagai memori *database* yang dapat meningkatkan performa *website*. penerapan *horizontal pod autoscaler* dan redis cluster mampu meningkatkan performa *website elearning moodle* sebesar 4,3% dibandingkan dengan pendekatan monolitik. Penelitian menunjukkan bahwa penerapan kubernetes dan *redis cluster* mampu meningkatkan performa *website elearning moodle*. Penelitian ini juga menunjukkan bahwa pendekatan *microservice* mempunyai performa yang lebih baik dibandingkan dengan pendekatan monolitik.

Kata kunci: horizontal pod autoscaler, kubernetes, redis, moodle, cloud computing

Abstract

Elearning is a very vital tool in learning today. To provide optimal service, elearning servers require quite large computing resources when a large number of users access them simultaneously. However, expensive computing resources such as CPU, memory and disk storage make it difficult for organizations to meet the needs of large users. Previous research compared the performance of two public clouds on a moodle-based learning management system. The results showed that backup and restore times increased by about 10 seconds for every additional 500 MB of data size. This research aims to apply Kubernetes-based horizontal pod autoscaler and Redis cluster on the Moodle elearning server. Moodle is used to run elearning and Redis as database memory which can improve website performance. Horizontal implementation of pod autoscaler and Redis cluster was able to increase the performance of the Moodle e-learning website by 4.3% compared to a monolithic approach. Research shows that implementing Kubernetes and Redis clusters can improve the performance of Moodle e-learning websites. This research also shows that the microservice approach has better performance compared to the monolithic approach.

***Keywords**: horizontal pod autoscaler, kubernetes, redis, moodle, cloud computing*

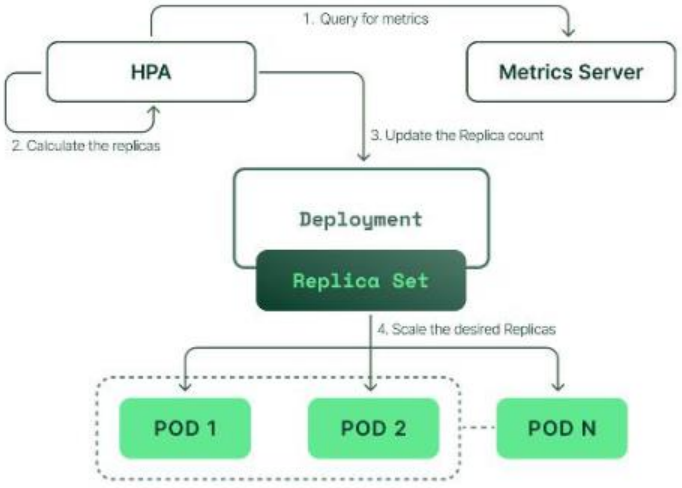
1. PENDAHULUAN

Komputasi awan merubah cara organisasi dalam mengelola sumber daya komputasinya. Komputasi awan menawarkan tiga jenis layanan, yaitu *Infrastruktur As A Service* (IAAS), *Software As A Service* (SAAS) dan *Platform As A Service* (PAAS). Ketiga jenis layanan tersebut diterapkan baik secara *private*, *public* maupun gabungan dari keduanya. IAAS menawarkan pengelolaan sumber daya komputasi secara terpusat oleh penyedia layanan, meliputi CPU, *memory*, *disk* penyimpanan, sistem operasi dan jaringan. Klien atau pengguna cukup membayar sumber daya komputasi yang digunakan saja. Banyak penyedia komputasi awan publik yang memberikan layanan *horizontal pod autoscaler* (HPA), namun tidak banyak referensi konfigurasi HPA pada pengelolaan komputasi awan privat atau *on premises*. Berlangganan pada penyedia komputasi awan publik juga membutuhkan biaya yang mahal.

Elearning merupakan sarana yang sangat vital dalam pembelajaran saat ini. Untuk memberikan layanan yang optimal, *server elearning* membutuhkan sumber daya komputasi yang cukup besar ketika *user* mengakses secara bersamaan dalam jumlah besar. Namun, mahalnya sumber daya komputasi seperti CPU, *memory* dan disk penyimpanan membuat organisasi kesulitan dalam memenuhi kebutuhan pengguna yang besar. Oleh karena itu sumber daya komputasi harus bisa secara dinamis menyesuaikan jumlah pengguna yang mengakses layanan elearning. Ketika jumlah pengguna *elearning* sedikit, secara otomatis sumber daya komputasi berkurang begitu juga ketika jumlah pengguna meningkat maka seacara otomatis sumber daya komputasinya juga meningkat. Komputasi awan khususnya IAAS menawarkan solusi ini dengan memanfaatkan teknologi docker dan kubernetes.

Teknologi kontainer memberikan elastisitas pengelolaan sumber daya komputasi. Kontainer adalah sebuah *lightweight virtual machine* yang mampu menjalankan aplikasi dengan pemakaian sumber daya komputasi yang kecil[1]. Salah satu teknologi kontainer yang populer adalah docker[2]. Untuk mengelola docker dengan mudah dapat menggunakan sebuah platform *open source*, yaitu kubernetes[3] yang dapat berjalan pada kluster publik maupun pada data *center* pribadi. Docker merupakan salah satu bentuk implementasi pendekatan berbasis *microservice*. Menurut [4] aplikasi berbasis monolitik memiliki performa yang lebih baik dibandingkan aplikasi dengan arsitektur *microservice*, namun aplikasi berbasis *microservice* memiliki kelebihan yaitu kemampuan isolasi yang lebih baik. Sedangkan menurut [5] konsumsi CPU aplikasi berbasis *microservice* lebih kecil dibandingkan konsumsi CPU aplikasi monolitik. Penelitian yang dilakukan oleh [6] menerapkan arsitektur *microservice* untuk resiliensi sistem informasi namun tidak mengukur performa arsitektur yang dibangun.

Teknologi kubernetes dipilih karena memiliki berbagai macam fitur, salah satunya adalah *Horizontal Pod Autoscaler* (HPA). HPA secara otomatis akan memperbanyak jumlah Pod di dalam ReplicationController, *Deployment*, ReplicaSet ataupun StatefulSet berdasarkan hasil observasi penggunaan CPU (atau, dengan metrik khusus, pada beberapa aplikasi yang menyediakan metrik). Perlu dicatat bahwa *Horizontal Pod Autoscale* tidak dapat diterapkan pada objek yang tidak dapat diperbanyak, seperti DeamonSets. *Horizontal Pod Autoscaler* diimplementasikan sebagai Kubernetes API *resource* dan sebuah *controller*. *Resource* tersebut akan menentukan perilaku dari *controller*-nya. *Controler* akan mengubah jumlah replika pada ReplicationController atau pada *Deployment* untuk menyesuaikan dengan hasil observasi rata-rata penggunaan CPU sesuai dengan yang ditentukan oleh pengguna[7]. Mekanisme Horizontal Pod Autoscaler pada Kubernetes ditunjukkan pada Gambar 1.



Gambar 1. Mekanisme Horizontal Pod Autoscaler pada Kubernetes[7]

Moodle adalah salah satu platform *elearning* yang paling populer digunakan di dunia pendidikan. *Elearning* berbasis Moodle mampu menangani *concurrent user* dalam jumlah ribuan

klien secara bersama-sama. Namun, pengelolaan sumber daya komputasi pada server *elearning* yang tidak secara dinamis menyesuaikan jumlah *concurrent user* mengakibatkan borosnya penggunaan CPU dan RAM. Untuk itulah diperlukan sebuah pengelolaan sumber daya komputasi agar mampu menyesuaikan jumlah *concurrent user*, ketika *concurrent user* sedikit maka secara otomatis CPU dan RAM berkurang dan sebaliknya saat *concurrent user* meningkat jumlah CPU dan *memory* meningkat.

Penelitian ini bertujuan menerapkan *horizontal pod autoscaler* dan *redis cluster* berbasis Kubernetes pada server *elearning* Moodle. Redis dipilih karena menurut [8] penggunaan redis dapat mempercepat akses data relational dalam aplikasi SIA dibandingkan melalui basis data Mysql secara langsung yang menggunakan *join query*. Namun [8] belum menerapkan arsitektur *microservice* dalam mengimplementasikan redis. Server *elearning* yang digunakan adalah server elearning Jurusan Teknologi Informasi Politeknik Negeri Jember. *Horizontal autoscaling* merupakan salah satu fitur yang disediakan oleh Kubernetes[9] yang menggunakan Docker sebagai lingkungan dasar untuk menjalankan kontainer yang portable dan mandiri. Kubernetes digunakan untuk mengelola *docker cluster* dan menjalankan ingress *controller* agar layanan *elearning* dapat diakses dari jaringan eksternal. *Elearning* berjalan pada pod yang dimonitor oleh Kubernetes, ketika jumlah *user* meningkat maka secara otomatis Kubernetes akan mereplikasi pod dan menambahkan ke dalam *service cluster*. Saat jumlah *user* menurun maka Kubernetes akan secara otomatis mematikan pod dari *service cluster*. Kubernetes Resource Metrics digunakan untuk mengetahui performa Kubernetes *cluster*.

Layanan *Horizontal Pod Autoscaler* pada umunya disediakan oleh penyedia *public cloud computing*, seperti Amazon Web Service (AWS), Alibaba Cloud, Google Kubernetes Engine (GKE). Namun untuk mendapatkan layanan tersebut kita harus membayar biaya langganan yang tidak murah sementara kita sudah mempunyai server sendiri untuk dikelola, sehingga diperlukan referensi konfigurasi HPA pada *private server*. Referensi mengenai konfigurasi HPA selama ini hanya terdapat pada penyedia *cloud computing public* dan belum ada pada *private cloud computing*. Kontribusi penelitian ini adalah konfigurasi *horizontal pod autoscaler* dan *redis cluster* berbasis Kubernetes pada server *private cloud computing*. *Elearning* berbasis moodle berjalan pada pod Kubernetes dan selanjutnya mengukur performa *horizontal pod autoscaler* dan *redis cluster*. Manfaat dari penelitian ini adalah meningkatkan *performance* dan *high availability* (HA) *elearning* berbasis moodle serta meningkatkan kepuasan pelanggan *elearning*.

2. METODE PENELITIAN

Metode penelitian yang digunakan dalam penelitian ini adalah modifikasi dari *network development life cycle* (NDLC) yang terdiri dari *Analysis*, *Design*, *Simulation Prototyping*, *Implementation*, *Monitoring* dan *Management*. Metode ini dipilih karena terbukti dapat meningkatkan fleksibilitas, skalabilitas dan efisiensi dalam pengembangan jaringan komputer[10]. Tahapan penelitian ditunjukkan pada Gambar 2. Tidak semua tahapan dalam NDLC dilakukan pada penelitian ini. Tahapan-tahapan yang dilakukan pada penelitian ini adalah *Analysis*, *Design*, *Implementation* dan *Evaluation*.

Gambar 2. Tahapan Penelitian

2.1. *Analysis*

Tahap ini merupakan tahap analisis kebutuhan, masalah, permintaan pengguna dan topologi jaringan. Metode yang dapat digunakan untuk tahap analisis antara lain: Analisis Kebutuhan, Analisis Masalah, Analisis Pengguna, Analisis Topologi.

2.2. *Design*

Pada tahap ini, data yang telah diambil akan menjadi dasar dalam pembuatan rancangan

topologi jaringan yang akan dibangun. Perancangan tersebut dapat berupa rancangan struktur topologi, rancangan tata letak kabel. Pada tahapan ini menggambarkan topologi jaringan yang digunakan sebagai objek penelitian yaitu Jurusan Teknologi Informasi Politeknik Negeri Jember.

2.3. *Implementation*

Pada tahapan ini mengimplementaikan kubernetes *cluster* pada data *center* dan selanjutnya melakukan performa *website elearning* menggunakan *benchmark tool* yang sudah tersedia pada Moodle. Jumlah pengguna bersamaan meningkat dari 100 sampai dengan 5000 *user* untuk membuktikan penambahan jumlah pod pada kluster kubernetes.

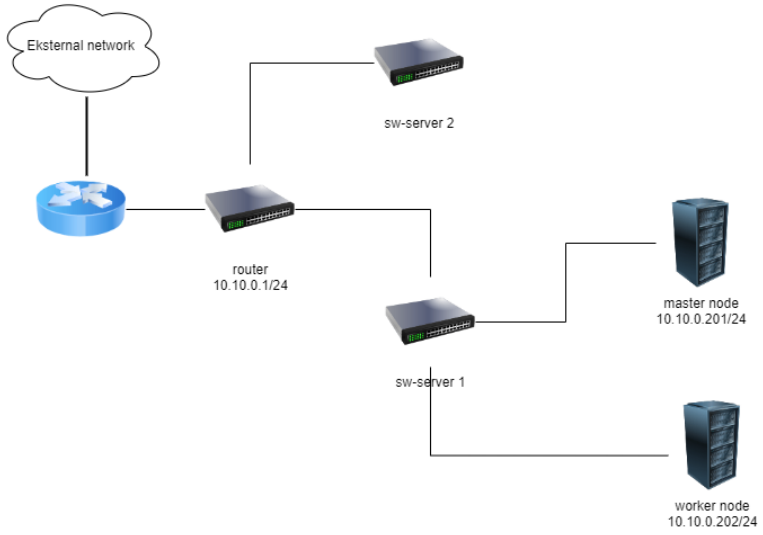
2.4. *Evaluation*

Pada tahapan ini dilakukan pemantauan pada kluster kubernetes yang menjalankan *elearning* untuk memastikan layanan berjalan dengan semestinya. *Evaluation* dilakukan dengan mengirimkan *traffic* ke *website elearning* yang berjalan pada kubernetes *cluster*. Ada tiga jenis penerapan *cloud computing* pada pengujian ini, yaitu monolitik dimana semua aplikasi diinstal pada *virtual maschine* yang sama, HPA tanpa *redis cluster* dan HPA menggunakan *redis cluster*. Selanjutnya diukur nilai benchmark menggunakan benchmark bawaan moodle. Hasil benchmark moodle dibandingkan antara server yang menerapkan pendekatan monolitik, *microservices* dan *microservice* yang dilengkapi dengan *redis cluster*.

3. HASIL DAN PEMBAHASAN

3.1 Hasil Penelitian

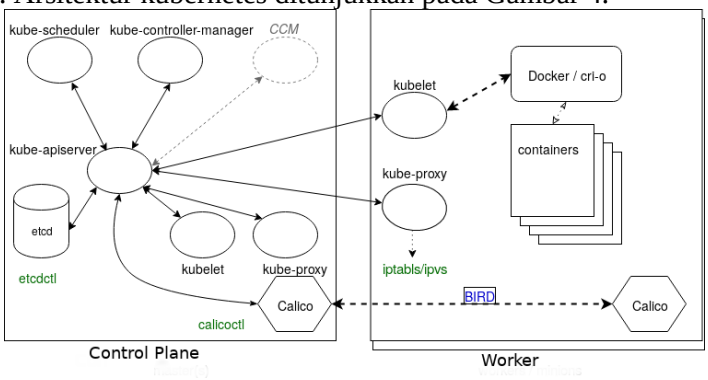
Penelitian ini dilaksanakan pada data *center* Jurusan teknologi Informasi Politeknik Negeri Jember. *Cluster* Kubernetes terdiri dari *node master* dan satu set *node worker*. Kubernetes Node Master menjalankan berbagai proses server dan pengelola untuk *cluster* dan *node worker* untuk tempat untuk pod yang didalamnya terdapat beberapa *container*. *Cluster* ini semua didorong melalui API call ke *operator*, baik lalu lintas interior maupun eksterior. Sebagian besar proses dijalankan di dalam *container*. Jumlah server fisik yang digunakan berjumlah 2 yang berfungsi sebagai master node dan *worker node*. Server yang digunakan memiliki spesifikasi CPU Intel Xeon RAM 8 GB disk 1 TB. Topologi jaringan ditunjukkan pada Gambar 3.



Gambar 3. Topologi Jaringan yang Digunakan Pada Penelitian

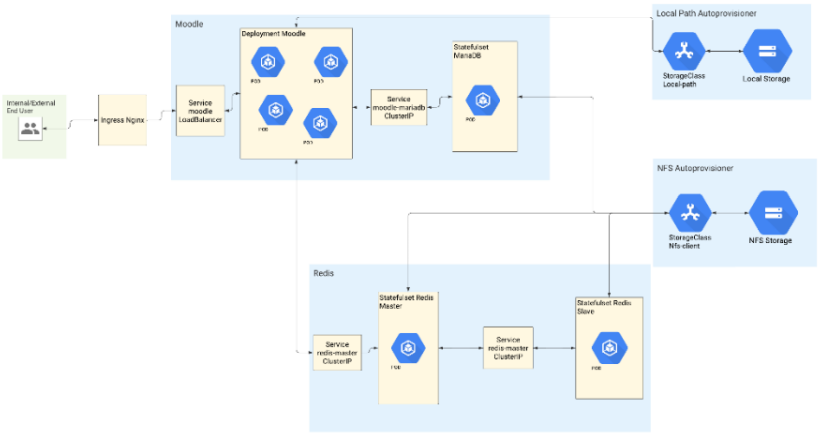
Cluster kubernetes terdiri dari master node yang berfungsi sebagai *control plane* dan *worker node* yang berfungsi sebagai *worker* tempat berjalanya pod. Pada penelitian ini menggunakan 1 server *control plane* dan 1 server *worker*. Pada *control plane* terdapat kube-apiserver, etcd, *kube-scheduler*, *kube-controller-manager*, kubelet, *kube-proxy*, *container*

runtime dan CNI. Kube-apiserver merupakan pusat dari operasi *cluster* kubernetes. Semua *request*/panggilan, baik trafik internal maupun eksternal, ditangani melalui *kube-apiserver*. *State cluster*, *network*, dan informasi persisten lainnya disimpan dalam DB etcd, atau lebih tepatnya penyimpanan b+*tree key-value*. *Kube-scheduler* menggunakan algoritma untuk menentukan node mana yang akan menampung pod. *Kube-controller-manager* adalah inti *control loop daemon* yang berinteraksi dengan *kube-apiserver* untuk menentukan status *cluster*. Kubelet merupakan agen yang dijalankan pada setiap *node* di *cluster* yang bertugas untuk memastikan *container* dijalankan di dalam Pod. *Kube-proxy* bertanggung jawab mengelola konektivitas jaringan ke *container*. *Container* runitime adalah perangkat lunak yang bertanggung jawab dalam menjalankan kontainer. CNI: adalah spesifikasi yang muncul dengan *library* terkait untuk menulis *plugin* yang mengonfigurasi *Container Network* dan menghapus *resource* yang dialokasikan saat *container* dihapus. Arsitektur kubernetes ditunjukkan pada Gambar 4.



Gambar 4. Arsitektur kubernetes

Gambar 5 menunjukkan gambaran sistem yang dikembangkan. *Cluster* kubernetes dibagi ke dalam 2 *namespace* yaitu moodle dan *redis*. Moodle digunakan untuk menjalankan *elearning* dan *redis* sebagai memori *database* yang dapat meningkatkan performa *website*[11]. Pada penelitian ini *redis* diimplementasikan ke dalam kubernetes *cluster*. Pada *namespace* moodle penulis men-deploy aplikasi moodle beserta database moodle, untuk menghubungkan jaringan moodle ke database mariadb, dihubungkan menggunakan *Service*. Sebuah *service* pada Kubernetes adalah sebuah abstraksi yang memberikan definisi set logis yang terdiri beberapa Pod serta *policy* bagaimana cara kamu mengakses sekumpulan Pod, seringkali disebut sebagai *microservices*. Set Pod yang dirujuk oleh suatu *Service* (biasanya) ditentukan oleh sebuah Label *Selector*.



Gambar 5. Gambaran Sistem yang Dikembangkan.

Pada penelitian ini penulis menggunakan *service* moodle-mariadb yang merujuk ke statefulset MariaDB *database* moodle. Statefulset adalah workload API objek yang digunakan untuk mengelola aplikasi stateful. Pod yang di-*deploy* menggunakan StatefulSet menggunakan

spesifikasi Pod yang sama. Untuk menghubungkan *deployment* moodle ke *service* moodlemariadb, perlu mendefinisikan *service* pada *deployment* moodle. *Deployment* adalah *controller* yang mengelola status ReplicaSet dan pod di dalamnya. *Control* tingkat yang lebih tinggi memungkinkan lebih banyak fleksibilitas dengan *upgrade* dan *administration*.

Selanjutnya agar *user* dapat mengakses moodle *e-learning* dengan menggunakan *service* LoadBalancer moodle. Yang nantinya *service* LoadBalancer akan mendefinisikan *port* sesuai yang tertera pada *list service*. Disini penulis masih mengakses menggunakan ip dan *port* dikarenakan tidak memiliki domain untuk dicantumkan di server jurusan teknologi informasi. Nantinya untuk mengakses melalui domain, terdapat konfigurasi tambahan yaitu menggunakan sebuah ingress. Selanjutnya men-konfigurasi moodle untuk dihubungkan ke *cluster redis*, untuk manajemen cache. Pada moodle terdapat pengaturan untuk menggunakan *redis* pada menu *caching*. masukkan IP *address* beserta *port* nya dari service *ClusterIP redis-master* pada *form* konfigurasi *redis* moodle atau dapat menggunakan domain dari *service redis master*. Pada tahapan ini *redis* juga di-*deploy*.

Untuk dapat mengimplementasikan *horizontal pod autoscaler* (HPA) dan *redis cluster* berbasis kubernetes terlebih dahulu harus melakukan instalasi Docker Engine, Kubernetes, dan Helm. Setelah instalasi docker pada kedua server selesai langkah selanjutnya adalah instalasi Kubernetes. Tahap berikutnya adalah instalasi helm untuk menjalankan moodle. Sebelum berpindah ke tahapan berikutnya harus dipastikan bahwa kubernetes dan helm sudah terinstal dengan sempurna.

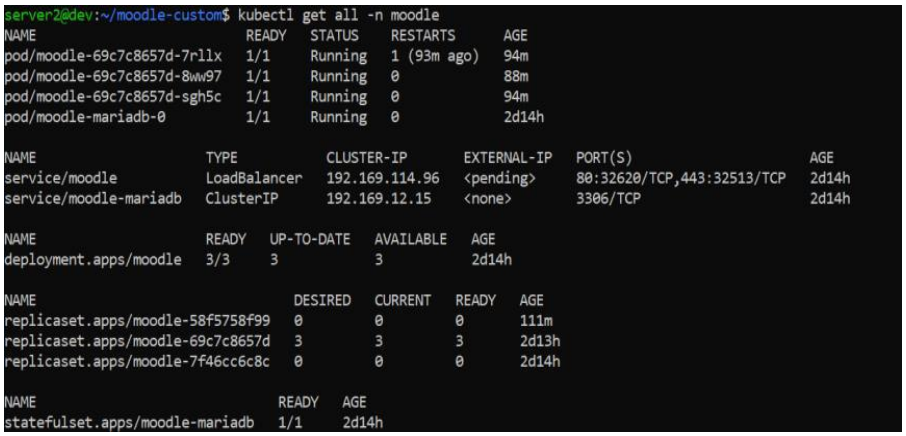
Tahapan berikutnya adalah instalasi *Host Path Storage* menggunakan Local Path Provisioner Rancher. *Host path storage* ini digunakan untuk menyimpan *file* agar tidak hilang ketika server dimatikan atau direstart. *Local path storage* ini juga berfungsi sebagai nfs server untuk menyimpan data-data *elearning* moodle. Langkah selanjutnya adalah kustom *docker image* moodle dan instalasi moodle *e-learning* menggunakan helm. Untuk melakukan kustom *docker image* moodle menggunakan Dockerfile yang ditunjukkan pada Gambar 6.

```
1)  FROM bitnami/moodle:latest
2)
3)  # Configure PHP error_log to stderr, as in:
4)  # https://docs.docker.com/config/containers/logging/
5)  RUN echo "error_log = /dev/stderr" > \
6)        /opt/bitnami/php/etc/conf.d/error_log.ini
7)
8)  # Prepare /var/local/cache
9)  RUN mkdir -vp /var/local/cache && \
10)     chown -Rc root:daemon /var/local/cache && \
11)     chmod ug+rwx /var/local/cache
12)
13) # Enable opcache
14) RUN echo "opcache.enable = 1" > \
15)       /opt/bitnami/php/etc/conf.d/opcache.ini
16)
17) # Prepare PECL build tools
18) RUN apt-get update && \
19)     apt-get install -y gcc make autoconf \
20)       libc-dev libssl-dev pkg-config
21)
22) # Install PECL igbinary
23) # igbinary serializer & lzf compression is recommended
24) RUN pecl channel-update pecl.php.net && \
25)     pecl install igbinary
26) RUN echo "extension=igbinary.so" > \
27)       /opt/bitnami/php/etc/conf.d/igbinary.ini
28)
29) # Install PECL redis
30) RUN pecl channel-update pecl.php.net
31) RUN yes | pecl bundle redis && cd redis && \
32)       phpize && ./configure --enable-redis-igbinary && \
33)       make && make install
34) RUN echo "extension=redis.so" > \
35)       /opt/bitnami/php/etc/conf.d/redis.ini
```

Gambar 6. *Dockerfile* untuk *Custom Image Moodle*

Untuk memastikan apakah *elearning* moodle sudah terinstal pada pod kubernetes, dapat

dilakukan pengecekan hasil dari *deploy* aplikasi moodle menggunakan helm. Berdasarkan gambar tersebut terdapat 4 pod yang berjalan dimana 3 pod menjalankan moodle dan 1 pod menjalankan maria-db. Moodle juga memiliki 2 *service* yaitu *service* dari moodle yang bertipe LoadBalancer dan *service* dari moodle-mariadb yang bertipe ClusterIP. ClusterIP adalah default, dan hanya menyediakan akses secara internal. Dapat dilihat pada gambar moodle memakai port 80 pada internal *cluster* dan diforwad ke port 32620 sehingga nantinya untuk bisa dibuka di *web browser* kita harus menggunakan http://<alamat ip server>:32620 atau dapat menggunakan *forwarding* port dari port 433 yaitu port 32513, pada *browser* menggunakan https://<alamat ip server>:32513. Pada *service* moodle *fitur load balancing* berjalan, *service* moodle membagi beban ke 3 pod yang sudah *ready* dan *running*, hal ini dapat dilihat pada *service* ketika menggunakan Kubernetes *dashboard*. Pengecekan hasil dari *deploy* aplikasi moodle menggunakan helm ditunjukkan pada Gambar 7.



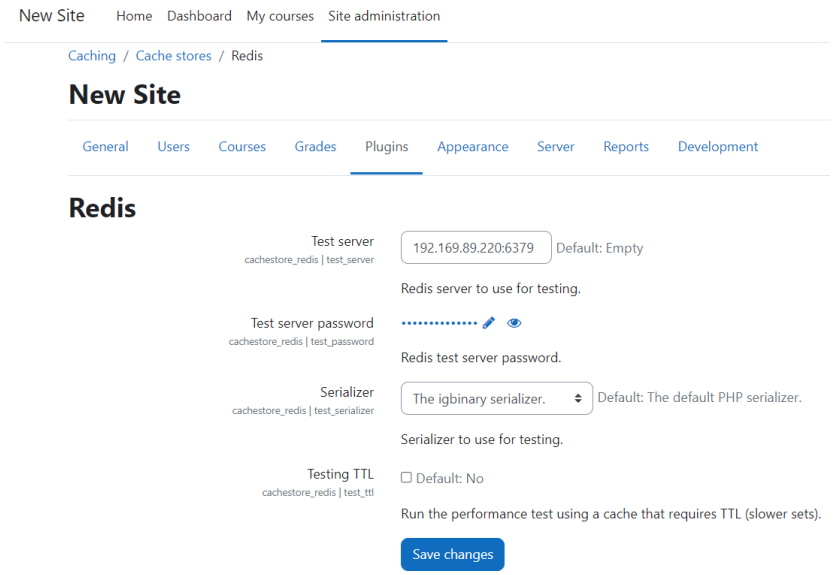
Gambar 7. *Resource Moodle*

Langkah selanjutnya instalasi *redis cluster* dan integrasi moodle dengan *redis cluster* untuk mengelola *cache*. Redis di *deploy* menggunakan *controller* statefulset sehingga setiap pod akan dianggap *unique*, dapat dilihat perbedaannya pada nama pod yang berakhiran angka. Angka dijadikan sebagai id dari pod. Pada redis terdapat 2 statefulset dan masing-masing memiliki 1 pod. Disini digunakan *redis cluster* sehingga terdapat *redis-master* dan *redis-slave*, *redis master* bersifat *read*/*write* dan *redis slave read-only*, sehingga untuk mengkoneksikan *redis* harus diarahkan ke *master*, apabila diarahkan ke *slave* maka akan terjadi *error* karena *slave* tidak punya hak akses untuk menulis data. Redis adalah aplikasi untuk mengelola *cache* pada aplikasi, data *cache* tersebut disimpan pada memori perangkat, jika perangkat mati maka data akan hilang. Pada *redis* ada opsi untuk membuat sebuah *cloning data* yang dapat disimpan pada *storage hard disk*, sehingga ketika perangkat mati data yang sebelumnya sudah disimpan dapat diambil lagi pada *storage*. Pada redis terdapat 3 *service* yaitu *service* untuk *master*, *headless*, dan *slave*. Gambar 8 menunjukkan *resource* pada *cluster redis*.

Gambar 8. Resource Pada Redis Cluster

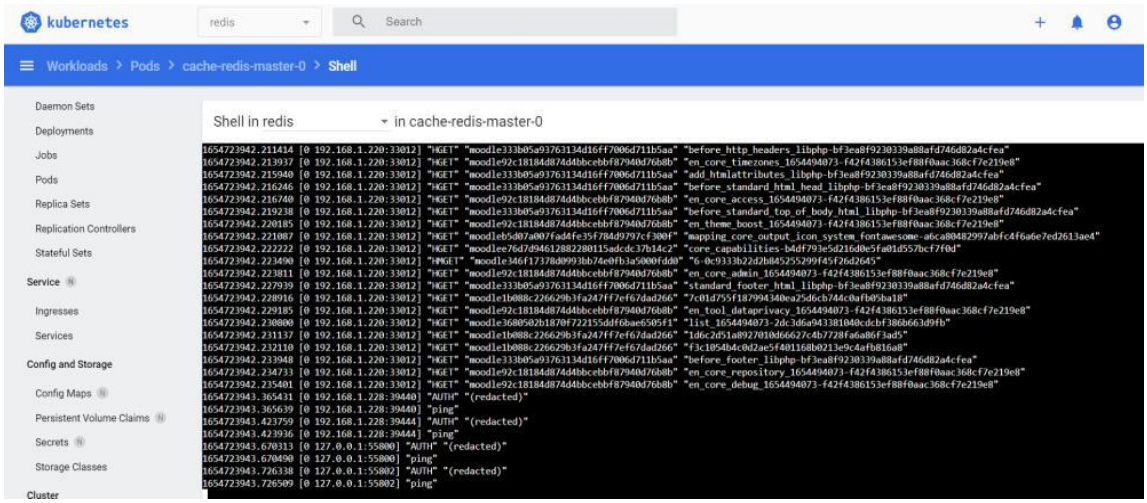
Langkah selanjutnya adalah penulis menempatkan *redis* untuk menyimpan *cache session* dan *application moodle*. *Cache session* disimpan di *redis* supaya ketika pod berkurang atau *down*, *user* yang sudah *login* tidak otomatis *logout* atau kembali ke *page login* karena *session* disimpan

pada *redis* sehingga meskipun pod berkurang atau *down session* tidak akan hilang. Untuk *application* digunakan untuk menyimpan data aplikasi, sehingga ketika ada *user* yang mengakses data yang sama, maka tidak perlu *query* ulang ke *database* cukup mengambil data dari *redis* sehingga tidak terlalu banyak beban untuk *query* ke *database*. Ketika data pada *database* diperbarui *redis* akan mengambil ulang data dari *database* untuk disimpan sehingga data juga ter-*update* sesuai yang ada pada *database*. Konfigurasi untuk menghubungkan *redis* ke moodle untuk menyimpan *cache session* dan *application* ditunjukkan pada Gambar 9.



Gambar 9. Konfigurasi Redis pada Website Moodle

Untuk memastikan apakah data dari moodle sudah masuk ke dalam *redis* perlu dilakukan pengecekan dengan *login* ke *redis master* melalui kubernetes *dashboard*. Pada Gambar 10 terlihat trafik yang masuk ke redis mempunyai *prefix* moodle. Hal ini berarti *implementation* instalasi *redis cluster* dan integrasi moodle dengan *redis cluster* untuk mengelola *cache* berhasil.



Gambar 10. Pengecekan koneksi moodle dan redis.

Website elearning moodle yang sudah terinstal pada *cluster* kubernetes hanya bisa diakses dari internal atau dari *master* dan *worker node* saja. Agar *website elearning* moodle dapat diakses dari eksternal maka harus menambahkan Ingress yang dalam hal ini menggunakan Nginx. Sebuah Ingress dapat dikonfigurasi agar berbagai *Service* memiliki URL yang dapat diakses dari eksternal

(luar klaster), melakukan *load balance* pada trafik, serta *Virtual Host* berbasis Nama. Sebuah kontroler Ingress bertanggung jawab untuk menjalankan fungsi Ingress yaitu sebagai *load balancer*, meskipun dapat juga digunakan untuk mengatur *edge router* atau *frontend* tambahan untuk menerima trafik. Gambar 11 menunjukkan pengecekan ingress *controller* telah berhasil dikonfigurasi dan *website elearning* dapat diakses dari eksternal menggunakan eksternal IP.

```
server2@server2:~$ kubectl get svc moodle -n moodle
NAME     TYPE           CLUSTER-IP        EXTERNAL-IP   PORT(S)                      AGE
moodle   LoadBalancer   192.169.140.170   10.10.0.201   80:32402/TCP,443:32361/TCP   7d8h
```

Gambar 11. Service ExternalIPs

Langkah selanjutnya adalah melakukan konfigurasi *horizontal pod autoscaler* (HPA) dengan membuat *file* manifest HPA. Konfigurasi HPA ditunjukkan pada Gambar 12. Pada gambar tersebut dapat diketahui minimal pod yang berjalan adalah 3 dan maksimal 14. Nilai maksimal ini ditentukan berdasarkan sumber daya yang dimiliki server fisik. Semakin besar sumber daya server fisik maka maksimal pod dapat bernilai semakin besar. Penambahan dan pengurangan pod berdasarkan besarnya penggunaan cpu dan *memory*. Rata-rata penggunaan cpu ditentukan sebesar 60%, jika pod menggunakan lebih dari 60% maka pod maka akan melakukan replica pod baru begitu juga sebaliknya jika sebuah pod menggunakan *resource* cpu kurang dari 60% maka akan dilakukan pengurangan replica.

```
1)  apiVersion: autoscaling/v2beta2
2)  kind: HorizontalPodAutoscaler
3)  metadata:
4)    name: moodle
5)    namespace: moodle
6)    labels:
7)      app.kubernetes.io/name: moodle
8)  spec:
9)    scaleTargetRef:
10)     apiVersion: apps/v1
11)     kind: Deployment
12)     name: moodle
13)   minReplicas: 3
14)   maxReplicas: 14
15)   metrics:
16)   - type: Resource
17)     resource:
18)       name: cpu
19)       target:
20)         type: Utilization
21)         averageUtilization: 60
22)   - type: Resource
23)     resource:
24)       name: memory
25)       target:
26)         type: AverageValue
27)         averageValue: 500Mi
28)   behavior:
29)     scaleDown:
30)       stabilizationWindowSeconds: 300
31)       policies:
32)       - type: Percent
33)         value: 10
34)         periodSeconds: 60
35)       selectPolicy: Min
36)     scaleUp:
37)       stabilizationWindowSeconds: 0
38)       policies:
39)       - type: Percent
40)         value: 60
41)         periodSeconds: 15
42)       - type: Pods
43)         value: 4
44)         periodSeconds: 15
45)       selectPolicy: Max
```

Gambar 12. File Manifest HPA

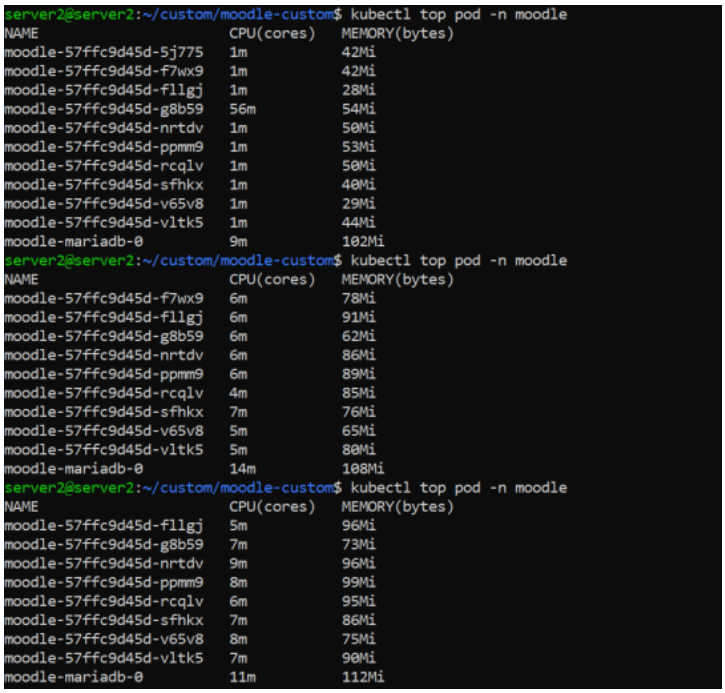
Tahap berikutnya adalah pengujian konfigurasi HPA dan *redis cluster* kubernetes. Pengujian dilakukan dengan mengirimkan trafik ke *website elearning* moodle menggunakan *software* Apache JMeter. Jumlah *user* meningkat dari 0, 1000, 2000, 3000, 4000 dan terakhir 20.000. Hasil *testing* menunjukkan pada bagian server node konsumsi memori dan cpu naik akan tetapi pada *testing* ke 3, konsumsi cpu server node tidak lebih besar dari *testing* sebelumnya, ada beberapa faktor yang mempengaruhi konsumsi node tidak lebih besar, yaitu bagian cpu naik ketika *scaling* up berjalan atau proses Pod baru dibuat, dan ketika pod sudah *running* konsumsi cpu berkurang. Untuk memori yang konsisten naik ketika *load testing* berjalan. Sehingga memori tetap konsisten naik meskipun pod sudah *running*. Hasil pengujian trafik ditunjukkan pada Tabel 1.

Tabel 1. Hasil Pengujian Trafik

Jumlah User	Jumlah Pod	Jumlah Memory Deployment	Jumlah CPU Deployment	Jumlah Memory Node	Jumlah CPU Node
0	3 pod	353 Mi	33 m	8 Gi	914 m
1.000	6 pod	1.260 Mi	1.484 m	8,5 Gi	2.254 m
2.000	7 pod	1.218 Mi	432 m	8,78 Gi	2.227 m

3.000	6 pod	2.133 Mi	343 m	9,2 Gi	1.748 m
4.000	6 pod	2.428 Mi	907 m	9,4 Gi	2.537 m
20.000	13 pod	1.974 Mi	3.702 m	8,8 Gi	6.095 m

Hasil *testing scaling down* ketika konsumsi *resource* turun dibawah parameter yang ditentukan, dan untuk *scaling down* diberi waktu untuk stabilisasi selamat 300 detik yang selanjutnya akan di *scaling down* 10% dari jumlah pod yang berjalan setiap 60 detik. Hasil pengamatan *scaling* down ditunjukkan pada Gambar 13. Berdasarkan gambar 13 dapat dilihat pada *list* pertama pod aplikasi moodle (Kecuali pod moodlemariadb) berjumlah 10, 60 detik selanjutnya menjadi 9 pod, dan 60 detik lagi menjadi 8 pod seterusnya hingga mencapai minimal pod yang sudah dikonfigurasi.



Gambar 13. Hasil Pengamatan *Scalling Down*

Pengujian benchmark moodle menunjukkan bahwa pada saat moodle diimplementasikan menggunakan pendekatan monolitik dimana moodle dan *database* diinstal pada *Virtual Machine* yang sama, didapatkan nilai benchmark moodle sebesar 93. Penerapan ini juga disebut dengan pendekatan monolitik dimana moodle dan database diinstal pada *Virtual Machine* (VM) yang sama. Setelah moodle diimplementasikan pada cluster kubernetes didapatkan nilai benchmark sebesar 91. Dan setelah moodle diimplementasikan pada *cluster* kubernetes dan *redis cluster*, didapatkan nilai benchmark sebesar 89. Nilai benchmark moodle semakin kecil menunjukkan performa *website* semakin baik. Hasil benchmark moodle ditunjukkan pada Tabel 2. Berdasarkan Tabel 2 dapat dikatakan bahwa penerapan *horizontal pod autoscaler* dan *redis cluster* mampu meningkatkan performa *website elearning* moodle sebesar 4,3% dibandingkan dengan pendekatan monolitik. Tabel 2 juga menunjukkan bahwa penerapan *microservice* pada *elearning* moodle mempunyai performa yang lebih baik dibandingkan dengan penerapan secara monolitik. Nilai benchmark moodle pada pendekatan secara *microservice* lebih baik dibandingkan dengan pendekatan secara monolitik. Selain itu juga penerapan redis cluster mampu meningkatkan performa *website elearning*.

Tabel 2. Hasil Pengukuran Benchmark Moodle

Pendekatan	***Horizontal Pod Autoscaler* (HPA)**	**Redis Cluster**	**Nilai Benchmark**
Monolitik	-	-	93

Cluster Kubernetes	yes	-	91
Cluster Kubernetes + redis Cluster	yes	yes	89

3.2 Pembahasan

Penelitian ini merupakan lanjutan dari penelitian yang dilakukan oleh [12] yang menerapkan *microservice* pada *Hotel Management System*. Namun pada penelitian tersebut belum menerapkan *autoscaling* sedangkan pada penelitian ini sudah menerapkan *autoscaling* dan juga *redis cluster*. Penelitian [12] menunjukkan bahwa penggunaan pendekatan monolitik lebih baik dibandingkan dengan penggunaan pendekatan *microservice*. Pada penelitian tersebut aplikasi *Hotel Management System* diimplementasikan menggunakan kontainer docker. Ada beberapa faktor yang menyebabkan perbedaan ini, yaitu jenis aplikasi yang digunakan untuk melakukan pengujian. Pada penelitian ini menggunakan *elearning* moodle sedangkan pada penelitian sebelumnya menggunakan *Hotel Management System*. Sedangkan penelitian yang dilakukan oleh [13]

Hasil penelitian ini juga sejalan dengan penelitian yang dilakukan oleh [11] yang menerapkan *redis* untuk meningkatkan performa *website*. Penelitian [11]menunjukkan bahwa penggunaan strategi *caching* baca/tulis dengan data serial juga telah meningkatkan kinerja aplikasi web. Permintaan untuk mengambil data dari cache dieksekusi lebih cepat karena data sudah diserialkan dan disimpan dalam memori, sehingga menghindari akses ke sumber data atau melakukan operasi yang rumit. Hal ini terutama berlaku ketika data yang jarang dimodifikasi digunakan dan salinan yang disimpan dapat digunakan untuk mempercepat akses. Selain itu, penggunaan strategi *caching* baca/tulis memberikan kemampuan untuk mengelola data dalam *cache* secara lebih fleksibel dan menjaga relevansinya. Hal ini dilakukan dengan mengatur masa pakai data yang di-*cache* atau secara eksplisit menghapus data dari *cache* untuk memperbaruinya dari sumber data [11].

Penelitian [14] membandingkan performa dua *public cloud* pada *learning management system* berbasis moodle. Hasilnya waktu *backup* dan *restore* akan meningkat sekitar 10 detik untuk setiap ukuran data tambahan sebesar 500 MB. Di sisi utilitas, penggunaan CPU saat *restore* membutuhkan lebih banyak daya daripada saat *backup*. Spike CPU maksimum sebesar 3,5% pada ukuran data 3 GB di kedua klaster *public cloud* saat *backup* dan *restore*. Namun pada penelitian tersebut tidak mengukur performa *autoscaling* dengan penambahan dan pengurang jumlah *user*. Perbedaanya lagi pada penelitian ini menggunakan *private cloud* sedangkan pada [14] menggunakan *public cloud*.

Penelitian [15] mengimplementasikan LMS moodle pada kontainer yang terinstal pada VPS server. Namun penelitian tersebut tidak mengukur performa LMS yang dijalankan sedangkan pada penelitian ini mengukur performa menggunakan *benchmark* bawaan moodle. Penelitian ini diharapkan dapat memberikan kontribusi berupa referensi penerapan kubernetes dan *redis cluster* LMS moodle pada *private cloud*. Penerapan kubernetes *cluster* pada LMS moodle didominasi oleh *public cloud provider* seperti yang dilakukan oleh [14].

4. KESIMPULAN DAN SARAN

Penelitian menunjukkan bahwa penerapan kubernetes dan *redis cluster* mampu meningkatkan performa *website elearning* moodle. Penelitian ini juga menunjukkan bahwa pendekatan *microservice* mempunyai performa yang lebih baik dibandingkan dengan pendekatan monolitik. Hasil penelitian ini juga membuktikan bahwa *cluster kubernetes* dan *redis* dapat diimplementasikan pada *private cloud* bukan hanya pada *public cloud provider* yang tentu saja harganya mahal. Penelitian selanjutnya dapat mengimplementasikan ingress *controller* selain nginx seperti yang digunakan pada penelitian ini dan membandingkan hasilnya.

DAFTAR PUSTAKA

[1] C. C. Chang, et al., "A Kubernetes-Based Monitoring Platform for Dynamic Cloud Resource Provisioning," *2017 IEEE Global Communications Conference, GLOBECOM 2017*, pp. 1–6, 2017.

[2] J. Shah and D. Dubaria, "Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform," *2019 IEEE 9th Annual Computing and Communication Workshop and Conference, CCWC*, pp. 184–189, 2019.

[3] L. A. Vayghan, et al., "Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned," *IEEE International Conference on Cloud Computing, CLOUD*, pp. 970–973, 2018.

[4] A. Barczak and M. Barczak, "Performance Comparison of Monolith and Microservices Based Applications," *Proceedings of the 25th World Multi-Conference on Systemics, Cybernetics and Informatics, WMSCI 2021*, pp. 120–125, 2021.

[5] F. Tapia, et al., "From Monolithic Systems to Microservices: A Comparative Study of Performance," *Applied Sciences (Switzerland)*, vol. 10, no. 17, 2020.

[6] H. Suryotrisongko, "Arsitektur Microservice untuk Resiliensi Sistem Informasi," *SISFO*, vol. 06, no. 02, pp. 231–246, 2017.

[7] M. Ahmed, "Kubernetes Autoscaling 101: Cluster Autoscaler, Horizontal Autoscaler, and Vertical Pod Autoscaler." 2019. Available: https://www.cncf.io/blog/2019/10/29/kubernetes-autoscaling-101-cluster-autoscaler-horizontal-autoscaler-and-vertical-pod-autoscaler/

[8] M. I. Zulfa, A. Fadli, and A. W. Wardhana, "Application Caching Strategy Based on in-Memory using Redis Server to Accelerate Relational Data Access," *Jurnal Teknologi dan Sistem Komputer*, vol. 8, no. 2, pp. 157–163, Apr. 2020.

[9] T. T. Nguyen, at al., "Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration," *Sensors*, vol. 20, no. 16, pp. 1–18, 2020.

[10] M. S. Bahry and B. Sugiantoro, "Analysys and Implementation IEEE 802.1q to Improve Network Security," *(IJID) International Journal on Informatics for Development*, vol. 6, no. 2, pp. 28–33, 2017.

[11] M. V. Privalov and M. V. Stupina, "Improving Web-Oriented Information Systems Efficiency using Redis Caching Mechanisms," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 33, no. 3, pp. 1667–1675, 2024.

[12] B. M. Susanto, E. S. J. Atmadji, and L. Hakim, "The Performance of Hotel Management System Using Microservices and Containerization Technology," in *Proceedings of the 4th International Conference on Social Science, Humanity and Public Health, ICoSHIP 2023*, 2023.

[13] O. Al-Debagy and P. Martinek, "A Comparative Review of Microservices and Monolithic Architectures," *2018 18th IEEE International Symposium on Computational Intelligence and Informatics, CINTI*, pp. 149–154, 2018.

[14] A. F. Ranunegoro, F. Dewanta, and B. Aditya, "Analisis Migrasi Data LMS Pada Klaster Kubernetes Antar-Public Cloud Menggunakan Backup dan Restore," *Jurnal Multinetics*, vol. 9, no. 1, pp. 71–78, 2023.

[15] O. Soufiane, et al., "Implementation of the Moodle E-learning Platform from Server Selection to Configuration," *GSC Advanced Engineering and Technology*, vol. 1, no. 1, pp. 016–025, 2021.