JSRCS Journal of Students' Research in Computer Science

e-ISSN: 2722 – 290X
Vol. 5 No. 2 (Nopember 2024), Page: 185 – 196
https://doi.org/10.31599/vr275f23
Available Online at http://ejurnal.ubharajaya.ac.id/index.php/JSRCS

Konversi ke *Microservices* Untuk Peningkatan Layanan Perpustakaan

Malikus Sumadyo[1] , Rahmadya Trias Handayanto[1,*], Ramdhani Setiawan[1]

* Korespondensi: e-mail: rahmadya.trias@gmail.com

[1] Teknik Komputer; Universitas Islam 45 Bekasi; Jl. Cut Meutia No. 83, Bekasi 17113; e-mail: malikus.sumadyo@gmail.com, rahmadya.trias@gmail.com, rsetiawan@gmail.com

Submitted : **25 September 2024**
Revised : **28 Oktober 2024**
Accepted : **20 November 2024**
Published : **30 November 2024**

Abstract

Most web-based applications are applications that provide services to users. The services themselves often change according to policies that are usually based on user satisfaction information. Unfortunately, existing applications are mostly based on open-source which is mostly monolithic. If there is an addition of a new service, it requires development from the beginning again. Especially when there is an imbalance between one service and another where one service requires large resources while other services do not require too many resources such as processors, network speed, or database size. This study tries to propose a system change in the library based on PHP-MySQL to be based on microservices.

Keywords: *Docker, Kong, Library, Microservices, Virtualization*

Abstrak

Sebagian besar aplikasi berbasis web adalah aplikasi yang memberikan pelayanan kepada pengguna. Pelayanan sendiri kerap berubah mengikuti kebijakan yang biasanya dari informasi kepuasan pengguna. Sayangnya aplikasi yang existing sebagian besar berdasarkan open-source yang sebagian besar monolitik. Jika ada penambahan layanan baru membutuhkan pengembangan dari awal lagi. Apalagi ketika adanya ketidakseimbangan antara satu layanan dengan layanan lainnya dimana satu layanan membutuhkan sumber daya yang besar sementara layanan lainnya tidak terlalu membutuhkan sumber daya misalnya prosesor, kecepatan network, maupun ukuran basis data. Penelitian ini mencoba mengusulkan perubahan sistem di perpustakaan yang berbasis *PHP-MySQL* menjadi berbasis *microservices*.

Kata Kunci: *Docker, Kong, Library, Microservices, Virtualisasi*

1. Pendahuluan

Peralihan dari proyek yang bersifat monolitik menjadi *microservices* membutuhkan beberapa paradigma baru dimana masalah kerap muncul ketika dikaitkan dengan manajemen proyek berbasis tahun anggaran, misalnya pada proyek pemerintah yang kerap dilaksanakan tiap tahunnya (Sumadyo et al., 2023). Ketika kontrak Kerangka Acuan Kerja (KAK) ditandatangani, manajer proyek harus mengikuti semua kebutuhan yang ada di KAK tersebut. Terkadang KAK yang dibuat oleh departemen pemberi kerja tidak menggambarkan seluruh kebutuhan yang seharusnya dibuat akibat dari berbedanya personil pembuat KAK dengan staf yang nantinya akan menggunakan aplikasi. Jika menggunakan aplikasi monolitik, biasanya muncul masalah ketika

ada perubahan tiba-tiba dari pengguna karena harus membongkar aplikasi yang sudah dibuat saat studi kelayakan atau proses pengujian. Berbeda dengan *microservices*, Ketika ada perubahan tiba-tiba, hanya layanan tertentu saja yang dirubah baik layanan baru maupun penambahan fitur-fitur tertentu dari suatu layanan. Beberapa aplikasi saat ini sudah mendukung pengembangan *microservices* (Bozan et al., 2021).

Penelitian terdahulu (Hui et al., 2025) mengenai arsitektur *microservices*, menyoroti pentingnya pengujian untuk memastikan kualitas dan keandalannya, mengingat kesalahan dalam *microservices* dapat menyebabkan dampak serius. Penelitian ini menganalisis 93 studi utama dari 19.595 penelitian terkait dan mengidentifikasi sembilan metode pengujian, empat tantangan utama, serta lima solusi yang diusulkan. Peneliti menemukan bahwa metode pengujian *microservices* berbeda signifikan dibandingkan pengujian arsitektur tradisional, terutama dalam konteks mekanisme komunikasi. Studi ini juga mengidentifikasi kesenjangan dalam metode pengujian yang ada dan menawarkan arahan penelitian di masa depan untuk memperkuat pengujian *microservices*.

Penelitian oleh (Hui et al., 2025) mengembangkan pendekatan inovatif untuk adaptasi global runtime pada aplikasi *microservices*, berdasarkan sintesis rekonfigurasi tingkat arsitektur. Peneliti merancang algoritma untuk skala otomatis proaktif-reaktif yang mencapai Beban Komputasi Maksimum sistem dengan melakukan orkestrasi penerapan yang optimal. Pendekatan ini dievaluasi menggunakan platform simulasi arsitektur *microservices* untuk membandingkan skala lokal/global dan reaktif/proaktif. Hasil benchmark empiris menunjukkan bahwa skala global proaktif secara konsisten mengungguli pendekatan reaktif. Namun, kinerja terbaik diperoleh dengan pendekatan unik yang menggabungkan proaktivitas dan reaktivitas. Pendekatan ini melampaui metode terkini ketika mempertimbangkan kinerja sekaligus efisiensi sumber daya.

Sementara itu, (Sellami & Saied, 2025) menunjukkan bahwa arsitektur *Microservice* muncul sebagai solusi untuk masalah yang sering terjadi dalam pengembangan dan pemeliharaan sistem monolitik, sehingga banyak perusahaan memigrasikan aplikasi besar mereka dari monolitik ke sistem berbasis *microservices*. Namun, penelitian tersebut mengungkapkan bahwa proses migrasi sangat mahal dan sulit, terutama pada tahap pembagian (*decomposition*) sistem monolitik. Pendekatan yang dapat mengotomatisasi dan membimbing pengembang pada tahap ini dapat membantu mengurangi kesulitan, tetapi penelitian lebih lanjut menunjukkan bahwa jumlah kombinasi potensi *microservices* meningkat secara eksponensial dengan skala sistem monolitik, sehingga menentukan struktur yang tepat menjadi tantangan karena kurang jelasnya kualitas yang diinginkan dari arsitektur baru. Untuk mengatasi masalah ini, penelitian mengusulkan pendekatan baru berbasis *Deep Reinforcement Learning (DRL)* yang disebut *RLDec*, yang dirancang untuk memberikan rekomendasi pembuatan *microservices* baru. Tugas ini diformulasikan sebagai *Markov Decision Process,* dan model *Deep Neural Network* dilatih menggunakan kualitas *microservices* yang dihasilkan sebagai fungsi *reward*, berdasarkan analisis struktur dan semantik dari sistem monolitik. Evaluasi dilakukan terhadap pendekatan

menggunakan berbagai metrik, eksperimen dalam beberapa skenario, perbandingan dengan metode pembagian lainnya, dan analisis mendalam dari contoh pembagian, dengan hasil detail yang disertakan dalam paket artefak penelitian.

Penelitian terdahulu (Zhang & Lu, 2023) menunjukkan bahwa WeChat Official Account (*WCOA*) telah menjadi *platform* yang penting dan inovatif dalam layanan perpustakaan akademik, yang secara signifikan mempercepat perkembangan di bidang ini. Sebuah studi menggunakan model yang diperbarui dari *DeLone and McLean (D&M)* serta metode *Delphi* berhasil mengembangkan sistem evaluasi kepuasan pengguna terhadap *Academic Library WeChat Official Account (ALWCOA)*. Penelitian tersebut melibatkan 212 mahasiswa universitas untuk mengisi kuesioner mengenai indikator evaluasi, yang kemudian dianalisis menggunakan *grey relational analysis (GRA)* dan prinsip Pareto. Hasilnya, tiga indikator utama yang memengaruhi kepuasan pengguna *ALWCOA* berhasil diidentifikasi, yaitu responsivitas layanan, ketepatan waktu informasi, dan keamanan sistem. Studi ini tidak hanya menawarkan strategi evaluasi yang efektif, tetapi juga memberikan wawasan untuk meningkatkan layanan *ALWCOA*.

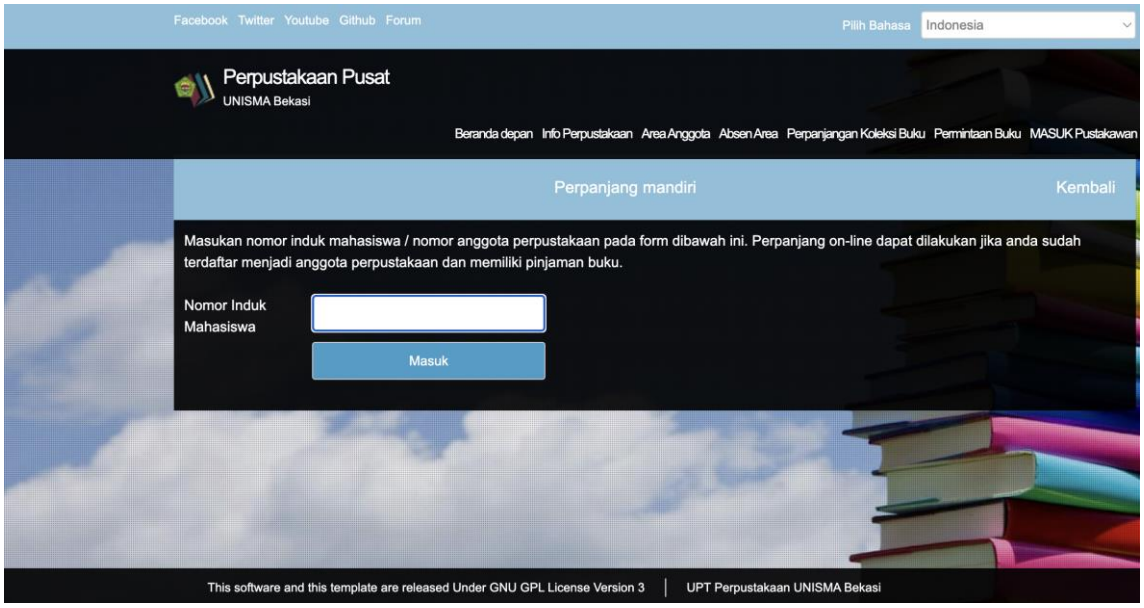
Penelitian terdahulu (Novytskyi, 2024) telah membahas pentingnya membangun lingkungan yang mendukung penelitian ilmiah, khususnya dalam konteks perkembangan *Open Science* di Ukraina. Penelitian ini melakukan tinjauan terhadap portal modern untuk agregasi data ilmiah, menganalisis alat-alat yang tersedia, serta mengidentifikasi kendala dalam pengumpulan data dari perpustakaan digital dan jurnal. Salah satu temuan penting adalah validasi penggunaan *VuFind* sebagai alat untuk menerapkan pendekatan *extraction–transformation–loading (ETL)* dalam agregasi data, yang memungkinkan standarisasi format dan nilai pada bidang metadata. Selain itu, penelitian ini menyoroti tantangan integrasi yang muncul akibat kurangnya persyaratan ketat pada struktur data dalam protokol OAI-PMH, meskipun protokol ini telah menjadi standar utama. Untuk mengatasi masalah ini, pendekatan ETL yang memanfaatkan metode ontologis seperti pemetaan data, linked data, dan kamus semantik diusulkan guna meningkatkan proses integrasi data. Penelitian ini juga meninjau potensi alat modern untuk integrasi OAI-PMH yang aktif dikembangkan, dengan tujuan mengembangkan protokol integrasi masa depan yang lebih sederhana dan mendukung validasi semantik data secara otomatis.

Penelitian yang menunjukan keunggulan *microservices* telah banyak dilakukan (Al Fath & Al Amin, 2022; Ferdinand et al., 2022; Nauli, 2017; Purwanto et al., 2021; Risnanto, 2022), dan saat ini sudah mulai merambah ke aplikasi perpustakaan. Oleh karena itu perlu implementasi *microservices* untuk aplikasi di perpustakaan. Di sini aplikasi dari perpustakaan Universitas Islam 45 yang bersifat monolitik akan dibuatkan konsep disain berbasis layanan dalam bentuk *microservices* (Gambar 1).

Sumber: https://perpustakaan.unismabekasi.ac.id/ (2024)

Gambar 1. Aplikasi Perpustakaan Universitas Islam 45

Website perpustakaan masih menggunakan model monolitik dengan bahasa PHP dan *database MySQL*. Kelemahan dari model monolitik adalah sulitnya menambahkan layanan-layanan baru karena harus mematikan seluruh layanan sebelumnya. Empat layanan yang akan dibuat mengikuti layanan pada moda monolitik antara lain: a) pendaftaran anggota, b) usulan pengadaan buku, c) perpanjangan peminjaman, dan d) survei pemustaka. Sementara layanan yang lain tidak membutuhkan sistem yang kompleks karena hanya menampilkan informasi, sehingga bisa hanya memanfaatkan kode *HTML* biasa. Gambar 2 memperlihatkan salah satu layanan dari aplikasi perpustakaan yang akan dijadikan salah satu servis pada *microservices*.



Sumber: http://uptperpustakaan.unismabekasi.ac.id/katalog-buku/index.php?p=perpanjang (2024)

Gambar 2. Layanan Perpanjangan Pinjaman

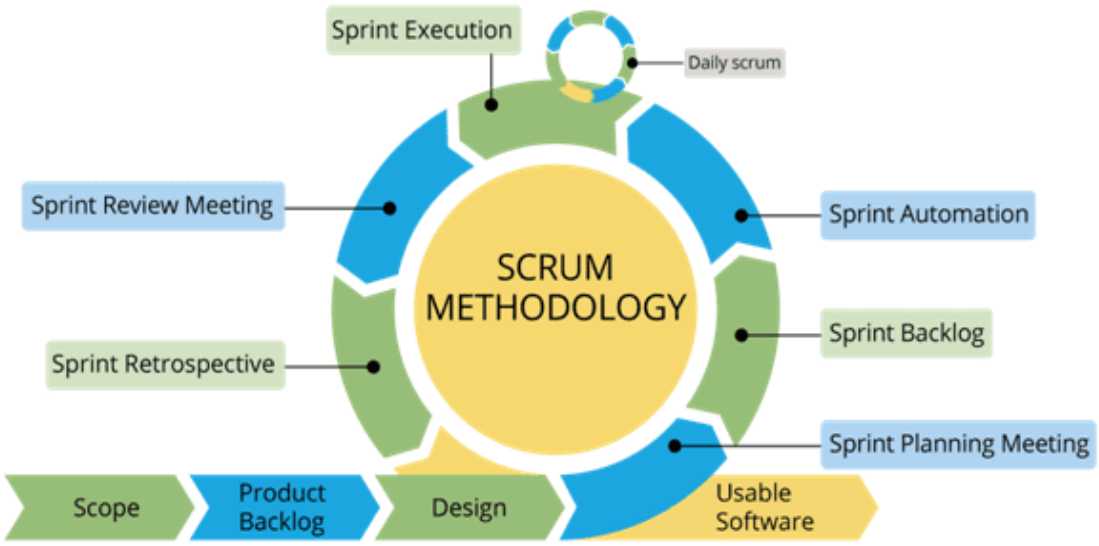
2. Metode Penelitian

Pengembangan sistem merupakan metode yang sangat menentukan keberhasilan proyek, khususnya yang melibatkan banyak layanan seperti *microsoervices*. Dalam penelitian metode *Agile* dengan *framework Scrum* dipilih. Metode ini dipilih karena sangat responsif terhadap permintaan dari pengguna. Saat ini terdapat enam metode yang dikenal sebagai metodologi *Agile*, yaitu *Crystal, Dynamic Software Development Method (DSDM), Feature-Driven Development (FDD), Lean Software Development, Scrum,* dan *Extreme Programming (XP).*

Kolaborasi dan komunikasi yang intens merupakan hal penting dalam metodologi *Scrum* (Sutherland, 2014). Tim pengembang bekerja sama dengan para pemangku kepentingan, seperti pengguna, pemilik produk, dan tim lainnya. Mereka berinteraksi secara terus-menerus untuk memahami perubahan kebutuhan dan memberikan umpan balik yang relevan. Hal ini memastikan bahwa perangkat lunak yang dikembangkan memenuhi harapan pengguna dan memberikan nilai yang optimal. Pengguna terkadang meminta perubahan tertentu dari KAK yang sudah disepakati bersama. Hal ini terjadi karena dalam pembuatan KAK pengguna kurang memiliki pengetahuan teknis, khususnya dalam teknologi informasi.

Model pengembangan harus mengacu pada *System Development Life Cycle (SDLC). Scrum* memiliki iterasi yang dikenal dengan istilah *increment* dimana tiap iterasi harus memiliki seluruh aspek dari *SDLC* mulai dari perencanaan, pemodelan, hingga implementasi. Dengan menerapkan metode ini, diharapkan pengembangan sistem dapat berlangsung dengan cepat, fleksibel, kolaboratif, dan efisien. Pendekatan memungkinkan pengembang untuk cepat menanggapi perubahan, bekerja sama dengan pihak terkait, dan menghasilkan perangkat lunak berkualitas yang sesuai dengan kebutuhan pengguna (Sutherland, 2014). Gambar 2 menunjukkan *framework Scrum* (Putri,2024)

.



Sumber: Putri (2024)

Gambar 3. Metode *Scrum*

Dalam *Scrum*, tahap *Scope* merujuk pada penentuan dan pengelolaan ruang lingkup (atau fitur) yang akan dikembangkan dalam sebuah proyek. Namun, dalam *Scrum*, penentuan ruang lingkup lebih bersifat dinamis dan iteratif, bukan tetap atau kaku. Ini karena *Scrum* menggunakan pendekatan yang berfokus pada iterasi (*Sprint*) untuk pengembangan dan memungkinkan ruang lingkup untuk berkembang atau disesuaikan seiring berjalannya waktu.

a. Product Backlog

Scope proyek secara keseluruhan awalnya ditentukan dalam Product *Backlog,* yaitu daftar prioritas fitur, perubahan, dan perbaikan yang dibutuhkan dalam produk. *Backlog* ini terus diperbarui berdasarkan umpan balik dan kebutuhan baru yang muncul.

b. Sprint Planning

Pada setiap *Sprint Planning meeting*, tim *Scrum* akan memilih item dari *Product Backlog* yang dapat diselesaikan dalam satu *Sprint*. Ini adalah ruang lingkup yang akan dikerjakan dalam *Sprint* tersebut, dan ini bisa berubah sesuai dengan perkembangan kebutuhan atau perubahan prioritas.

c. *Scope Creep*

Dalam *Scrum*, ada juga perhatian terhadap *Scope creep*, yaitu penambahan fitur atau perubahan yang tidak direncanakan di tengah jalan. Tim *Scrum* berusaha menghindari *Scope* creep yang tidak terkendali, dengan menjaga komunikasi yang jelas antara *Product Owner*, tim pengembang, dan pemangku kepentingan.

d. Increment

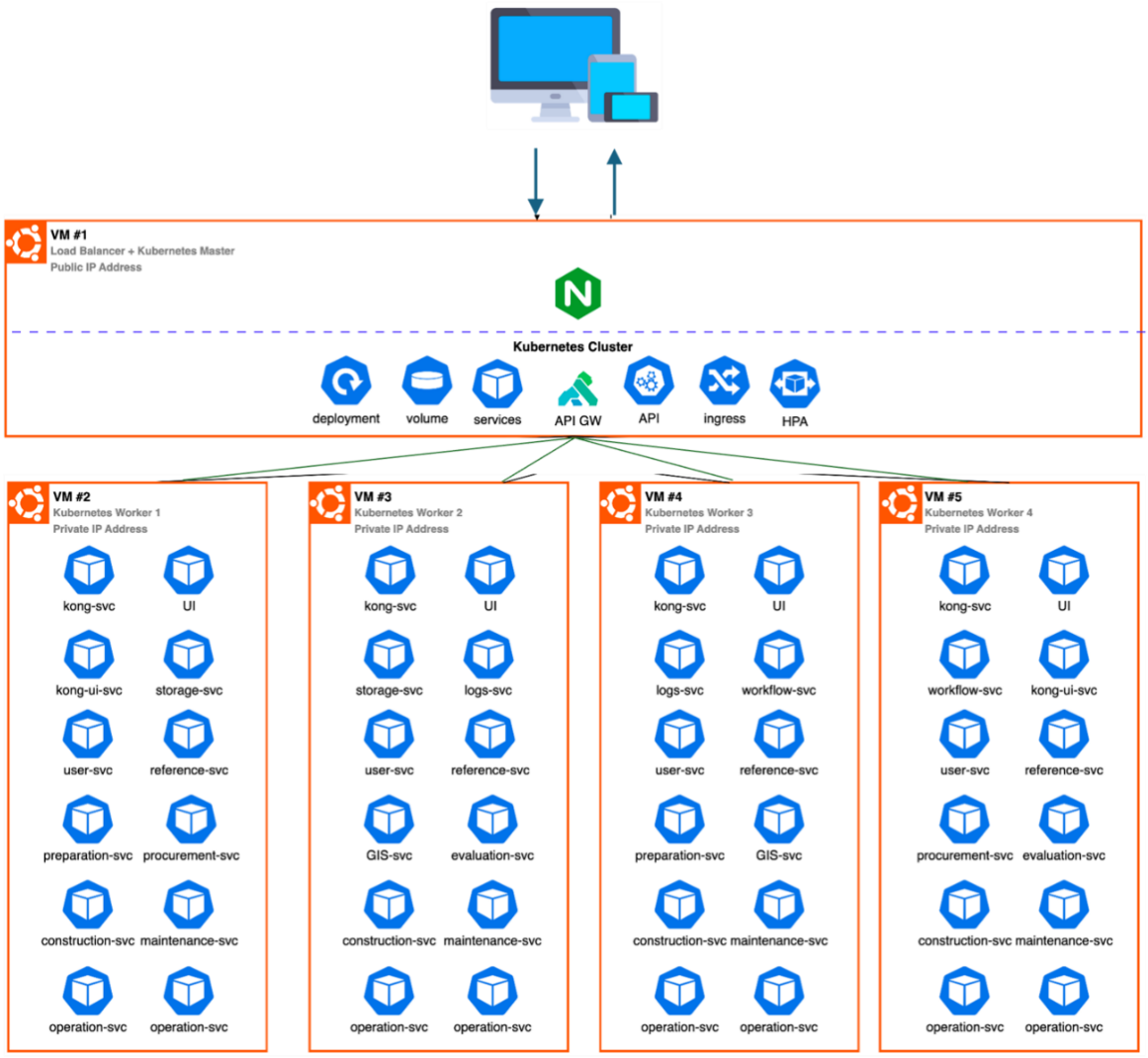
Hasil dari setiap *Sprint* adalah *Increment* yang berfungsi sebagai pengembangan produk secara bertahap. Pada setiap *Sprint*, ruang lingkup yang dikerjakan selesai dan dapat diterima, kemudian dilanjutkan dengan pengembangan lebih lanjut pada *Sprint* berikutnya.

Secara keseluruhan, dalam *Scrum*, *Scope* bersifat fleksibel dan terus berkembang, dan tim berfokus pada pengiriman hasil yang bernilai secara bertahap, sesuai dengan kebutuhan dan prioritas yang terus berubah.

3. Hasil dan Pembahasan

Tahap disain merupakan krusial karena tahap ini merupakan dasar dari tiap iterasi (*Sprint*) dalam model *Scrum*. Tiap iterasi akan menghasilkan satu layanan tertentu. Gambar 3 memperlihatkan disain *microservices* yang akan diterapkan. Tiap *virtual machine (VM)* dapat dibuat dalam satu atau beberapa *Sprint*.

Pengguna baik dengan laptop, tablet, maupun smartphone dapat mengakses aplikasi lewat mekanisme *request* dan *response*. *Virtual Machine* 1 akan berfungsi sebagai pengendali mesin virtual lainnya.



Sumber: Hasil Penelitian (2024)

Gambar 4. Disain *Microservices* Perpustakaan Universitas Islam 45

Kubernetes *cluster, API Gateway, Ingress, dan Horizontal Pod Autoscaler (HPA)* adalah komponen yang saling berkaitan dalam mengelola dan menjalankan aplikasi berbasis kontainer di Kubernetes, dengan hubungan sebagai berikut:

a. *Kubernetes Cluster*

Kubernetes *cluster* adalah sekumpulan *Node* yang menjalankan aplikasi yang telah dikemas dalam bentuk container. *Cluster* ini bertanggung jawab untuk menjadwalkan *workload* (*pods*) pada *Node*, mengelola scaling, pengaturan, serta komunikasi antar komponen. Selain itu, Kubernetes menyediakan berbagai fitur penting seperti *auto-scaling, self-healing,* dan *rolling updates*, yang memastikan aplikasi berjalan dengan efisien dan handal.

b. *API Gateway*

API Gateway adalah layanan yang berperan sebagai entry point untuk aplikasi *backend* yang berjalan di Kubernetes. Layanan ini memiliki fungsi utama, yaitu merutekan permintaan dari klien ke *service* tertentu di dalam *cluster*, menyediakan fitur autentikasi, otorisasi, *logging*,

serta transformasi permintaan dan respons. Contoh *API Gateway* yang umum digunakan antara lain Istio, Kong, dan AWS *API Gateway*. Dalam konteks Kubernetes, *API Gateway* sering bekerja bersama Ingress untuk mengatur lalu lintas ke layanan yang di-*host* di dalam *cluster*.

c. *Ingress*

Ingress adalah *resource* Kubernetes yang digunakan untuk mengelola lalu lintas *HTTP/HTTPS* dari luar ke dalam *cluster*. Fungsinya meliputi mendefinisikan aturan routing berdasarkan domain atau path URL ke service Kubernetes tertentu, mendukung SSL/TLS untuk komunikasi yang aman, serta sering diimplementasikan dengan controller seperti NGINX Ingress Controller atau Traefik. Selain itu, jika menggunakan *API Gateway*, Ingress dapat berperan sebagai "perpanjangan" dari gateway tersebut di tingkat *cluster*.

d. *Horizontal Pod Autoscaler (HPA)*

HPA (Horizontal Pod Autoscaler) adalah fitur Kubernetes yang digunakan untuk menskalakan jumlah pod secara otomatis berdasarkan beban kerja. Fungsinya meliputi menambah atau mengurangi jumlah pod berdasarkan metrik seperti penggunaan CPU, RAM, atau metrik kustom lainnya. HPA membantu memastikan aplikasi tetap responsif saat menghadapi beban tinggi sekaligus mengoptimalkan penggunaan sumber daya saat beban rendah. HPA bekerja bersama API Server untuk memantau metrik (melalui Metrics Server) dan menyesuaikan jumlah pod sesuai kebutuhan.

e. Hubungan Antar Komponen

API Gateway, Ingress, HPA, dan Kubernetes *cluster* saling bekerja sama untuk mengelola aplikasi dengan efisien. *API Gateway* menerima permintaan dari luar dan dapat meneruskannya ke Ingress, yang mendistribusikan lalu lintas lebih lanjut ke layanan di dalam *cluster* melalui aturan routing. Ingress sendiri merupakan bagian dari konfigurasi Kubernetes *cluster* yang mengatur lalu lintas eksternal ke internal. Di dalam *cluster*, HPA memonitor workload untuk memastikan aplikasi mampu menangani permintaan, termasuk yang masuk melalui *API Gateway* atau Ingress. Kubernetes *cluster* berfungsi sebagai fondasi yang menyediakan lingkungan runtime bagi aplikasi, layanan, dan pengelolaan otomatis seperti scaling melalui HPA.

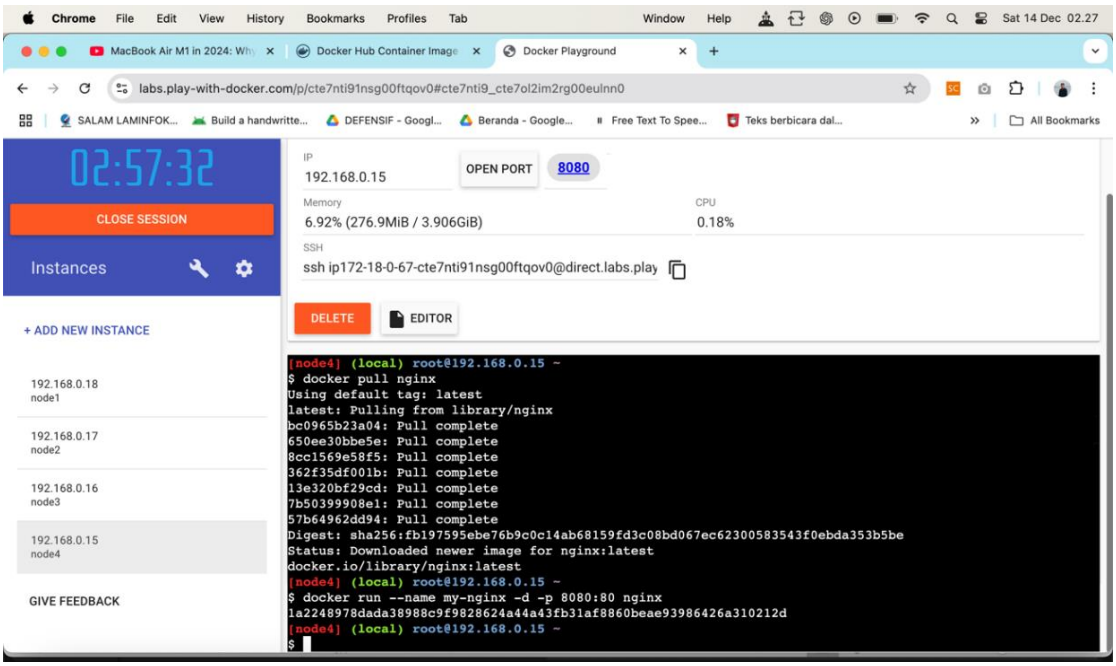
Klien mengirimkan permintaan HTTP ke *API Gateway*, yang kemudian diteruskan ke Ingress di *cluster* Kubernetes. Ingress mendistribusikan permintaan tersebut ke service yang sesuai. Apabila terjadi lonjakan beban, Horizontal Pod Autoscaler (HPA) akan meningkatkan jumlah pod aplikasi untuk memenuhi permintaan tersebut.

f. *Storage SVC*

Dalam konteks *microservices*, istilah *storage service* (atau *storage svc*) mengacu pada salah satu layanan yang secara khusus bertanggung jawab untuk mengelola penyimpanan data. Ini adalah bagian dari pendekatan arsitektur di mana setiap *microservice* memiliki data *independence* dan mengelola datanya sendiri.

Testing menggunakan *Docker* untuk memastikan lingkungan *system* dapat bekerjasama dengan baik. Tiap *virtual machine* memiliki *database* tersendiri. Namun untuk implementasi perlu diuji lebih lanjut terhadap *load balancing* Ketika menghadapi beban puncak, biasanya terjadi saat akhir-akhir batas yudisium dimana mahasiswa harus mengurus ke perpustakaan. Tiap layanan dibuatkan satu atau dua *Node* yang berisi server virtual dengan *Docker* di dalamnya. *Docker* mempermudah *deployment* jika sudah dibuat *image*-nya di server development.

Untuk pertukaran informasi antara satu *Node* dengan *Node* lainnya melalui mekanisme sharing *Application Programming Interface (API)*. *API* tersebut bisa berupa aplikasi atau akses ke data lewat mekanisme *Create, Read, Update,* dan *Delete*. Pilihan *database* antara lain *MySQL* dan *PosgreSQL*.



Sumber: Hasil Penelitian (2024)

Gambar 5. Testing dengan *Docker*

Setelah *microservices* dibuat dengan simulasi *Play with Docker*, selanjutnya tiap *image* disimpan dalam *hub Docker* (https://hub.*Docker*.com/). Untuk melakukan *push Docker image*, pertama login ke *Docker Hub*, beri tag pada *image* dengan format *username/repository:tag*, lalu gunakan perintah *Docker push username/repository:tag* untuk mengupload *image* ke *Docker Hub*. Manfaat dari dibuatnya *image* adalah developer mudah dalam memasang aplikasi dalam server *production*. Langkah proses *deployment* adalah dengan terlebih dahulu menyiapkan beberapa *Node* yang sudah diinstal *Docker* di dalamnya. Selanjutnya dengan menarik *Docker* dari *Docker hub* lewat perintah: *Docker pull username/imagename:tag* maka *image* akan ditarik ke *Node* tujuan. Di sini *username/imagename:tag* merupakan nama *hub Docker* (Gambar 6).

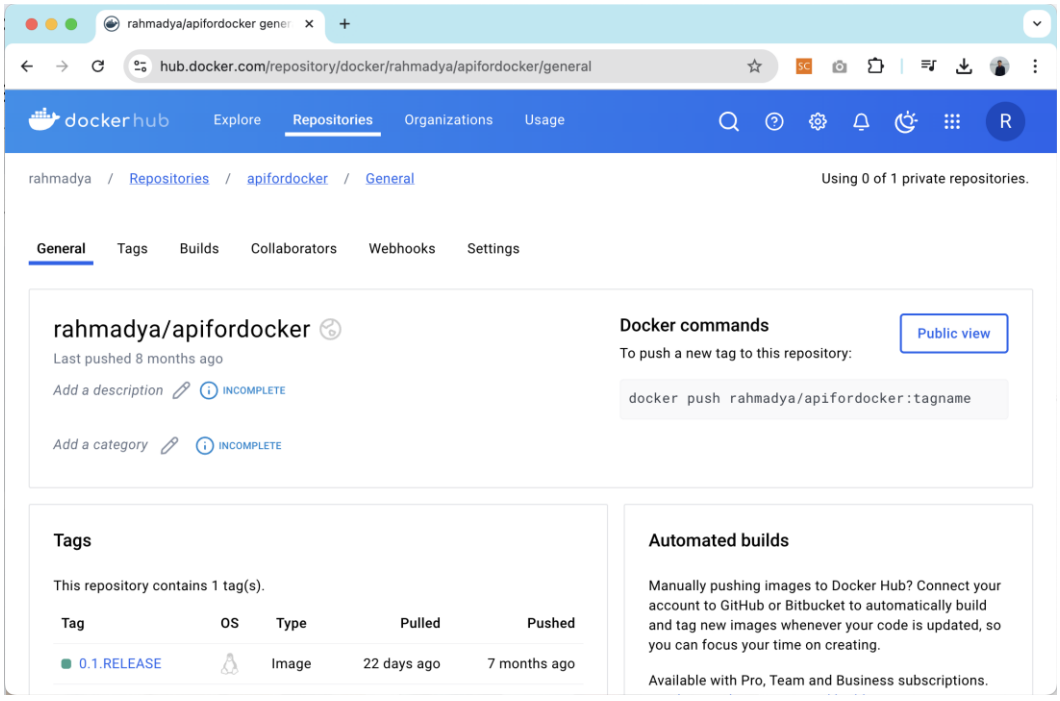
Cara lain selain dengan *hub Docker* adalah dengan *file image* yang dikirim ke *server* yang akan dipasang *Node* untuk layanan *microservices*-nya. Untuk memasang *file image Docker*

dengan nama *myimage.tar* ke *Docker* di mesin tujuan, langkah pertama adalah memastikan *Docker* sudah terinstal dan berjalan di mesin tersebut.

File *image* dapat diimpor menggunakan perintah *Docker load < myimage.tar*, yang akan menambahkan *image* tersebut ke daftar *image* lokal *Docker*. Setelah proses selesai, *image* dapat diverifikasi dengan perintah *Docker images* untuk memastikan *image* telah terdaftar. Selanjutnya, untuk menjalankan *image* ini sebagai *container*, gunakan perintah *Docker* run, misalnya *Docker run -d --name mycontainer -p 8080:80 myimage*, yang akan menjalankan container di latar belakang dengan nama "*mycontainer*" dan memetakan port 80 di dalam *container* ke port 8080 di mesin host. Setelah container berjalan, aplikasi di dalamnya dapat diakses melalui alamat *http://localhost:8080*, atau melalui port yang telah dikonfigurasi.

Untuk membuat API *CRUD (Create, Read, Update, Delete)* pada tabel anggota menggunakan PHP, langkah pertama adalah menyiapkan koneksi *database* menggunakan *PDO* atau *MySQLi*, yang akan digunakan untuk berinteraksi dengan *database*. Setelah itu, API diimplementasikan dalam beberapa endpoint, masing-masing untuk operasi *CRUD*. Setiap endpoint menerima request *HTTP* seperti *POST* untuk menambah data, *GET* untuk mengambil data, PUT untuk memperbarui data, dan *DELETE* untuk menghapus data anggota dari tabel.

Pada proses ini, *PHP* akan memproses permintaan tersebut dan menjalankan query *SQL* yang sesuai, misalnya *INSERT* untuk menambah data, *SELECT* untuk mengambil data, *UPDATE* untuk mengubah data, dan *DELETE* untuk menghapus data. Hasil dari query *SQL* kemudian dikirimkan kembali dalam format *JSON*, yang memudahkan aplikasi klien untuk memproses dan menampilkan data. Pastikan untuk selalu menangani input dengan aman dan menggunakan *prepared statements* untuk menghindari risiko *SQL injection*.



Sumber: Hasil Penelitian (2024)

Gambar 6. Hub *Docker* untuk *Deployment Image*

4. Kesimpulan

Aplikasi yang berkembang sesuai kebutuhan dapat memberikan manfaat lebih karena siklus SDLC yang berjalan dengan baik. Untuk departemen yang berkembang terus perlu memanfaatkan teknologi yang ada pada saat ini, salah satunya adalah *microservices*. Teknologi ini memanfaatkan virtualisasi berbasis layanan. Hasil eksperimen menunjukan dengan *microservices*, suatu layanan tidak saling tergantung dengan lainnya (*less couple*) sehingga mengurangi resiko berhentinya seluruh layanan, dalam hal ini perpustakaan Universitas Islam 45. Empat layanan yang akan dibuat mengikuti layanan pada moda monolitik antara lain: pendaftaran anggota, usulan pengadaan buku, perpanjangan peminjaman, dan survei pemustaka. Untuk implementasi diperlukan server berbasis linux dengan *Docker* yang sudah terinstal sebelum dipasang *image Docker* yang sudah dibuat di *server development*. Tiap layanan tersebut dibuat dalam satu atau dua *Node* di server. Komunikasi antar satu *Node* dengan lainnya melalui mekanisme *Application Programming Interface (API)*. Ke depan perlu memanfaatkan teknologi terkini yang digunakan untuk *full stack development* dengan kombinasi *front-end* dan *back-end* yang tepat.

Daftar Pustaka

Al Fath, T. R., & Al Amin, I. H. (2022). Implementasi Arsitektur *Microservice*s menggunakan RESTful API untuk Website Online Course Esploor. *Jurnal Teknik Informatika Unika ST. Thomas (JTIUST)*, *07*, 2657–1501.

Bozan, K., Lyytinen, K., & Rose, G. M. (2021). How to transition incrementally to *microservice* architecture. *Communications of the ACM*, *64*(1), 79–85. https://doi.org/10.1145/3378064

Ferdinand, J., Syahrina, A., & Musnansyah, A. (2022). Perancangan Arsitektur Perangkat Lunak *Microservice*s Pada Aplikasi Open Library Universitas Telkom Menggunakan gRPC. *Telkatika*, *1*(2), 71–77.

Hui, M., Wang, L., Li, H., Yang, R., Song, Y., Zhuang, H., Cui, D., & Li, Q. (2025). Unveiling the *microservice*s testing methods, challenges, solutions, and solutions gaps: A systematic mapping study. *Journal of Systems and Software*, *220*, 112232. https://doi.org/10.1016/j.jss.2024.112232

Nauli, S. B. (2017). Perancangan Aplikasi Peminjaman Buku Dengan Menggunakan Arsitektur *Microservice*. *Jurnal Ilmiah Fakultas Teknik LIMIT'S*, *13*(1). chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://teknik.usni.ac.id/jurnal/Sukarno B.pdf

Novytskyi, O. (2024). Metadata harvesting for digital library integration in Ukraine: a comparative study of the OAI-PMH protocol and VuFind's efficacy. *Digital Library Perspectives*. https://doi.org/10.1108/DLP-03-2024-0034

Purwanto, D., Pramusinto, W., & Utama, G. (2021). Aplikasi Kursus Online Berbasis Web Service Menggunakan Arsitektur *Microservice*s. *Proceeding SENDI_U*, 978–979.

Putri, N. A. (2024). *Scrum Prinsip Agile dan Tahapan Dalam Metode Scrum*. Telkom University. https://bif.telkomuniversity.ac.id/Scrum-prinsip-agile-dan-tahapan-dalam-metode-Scrum/

Risnanto, H. (2022). Rancang Bangun Sistem Informasi Layanan Mandiri Perpustakaan Berbasis Arsitektur *Microservice*. *Repository.Uinjkt.Ac.Id*.

Sellami, K., & Saied, M. A. (2025). Extracting *microservice*s from monolithic systems using deep reinforcement learning. *Empirical Software Engineering*, *30*(1). https://doi.org/10.1007/s10664-024-10547-4

Sumadyo, M., Handayanto, R. T., Whidhiasih, R. N., & Ekawati, I. (2023). The new framework for transition to *microservice*s on information technology project. *AIP Conference Proceedings*, *2706*(1), 20139. https://doi.org/10.1063/5.0120836

Sutherland, J. (2014). *More Praise for Scrum: The Art of Doing Twice the Work in Half the Time*.

Zhang, K., & Lu, P. (2023). What are the key indicators for evaluating the service satisfaction of WeChat official accounts in Chinese academic libraries? *Library Hi Tech*, *41*(3), 788–806. https://doi.org/10.1108/LHT-07-2021-0218