

BAB II

LANDASAN TEORI

2.1 Kajian Teori

Pada sub-bab ini akan diuraikan mengenai teori-teori dasar serta konsep ilmiah yang menjadi landasan dalam pelaksanaan penelitian. Teori yang dibahas mencakup teknologi komputasi awan (*cloud computing*), konsep kontainerisasi menggunakan *Docker*, hingga penggunaan berbagai pustaka dan kerangka kerja seperti Next.js, NestJS, dan Prisma ORM yang digunakan untuk membangun sistem otomatisasi *cloud hosting*. Selain itu, bagian ini juga menjelaskan metodologi pengembangan perangkat lunak yang diterapkan guna menjamin siklus pengembangan sistem yang terstruktur.

2.1.1 Cloud Computing

Cloud computing merupakan paradigma komputasi yang menyediakan berbagai sumber daya teknologi informasi melalui jaringan internet sehingga dapat diakses sesuai kebutuhan pengguna. Layanan yang disediakan meliputi server, media penyimpanan, basis data, jaringan, perangkat lunak, hingga layanan analitik tanpa mengharuskan pengguna memiliki maupun mengelola infrastruktur fisik secara langsung. Menurut *National Institute of Standards and Technology* (NIST), implementasi *cloud computing* memiliki lima karakteristik utama, yaitu *on-demand self-service*, *broad network access*, *resource pooling*, *rapid elasticity*, dan *measured service*, yang menjadi dasar dalam penyediaan layanan komputasi berbasis awan (Novianti Indah Putri et al., 2022).

Pada penelitian ini, model layanan yang diterapkan mengacu pada *Infrastructure as a Service* (IaaS) karena menyediakan infrastruktur komputasi yang dapat digunakan sesuai kebutuhan tanpa memerlukan pengelolaan perangkat keras secara langsung. Melalui model tersebut, pengguna memperoleh akses terhadap sumber daya seperti server virtual, jaringan, dan media penyimpanan untuk mendukung proses penyediaan layanan aplikasi.

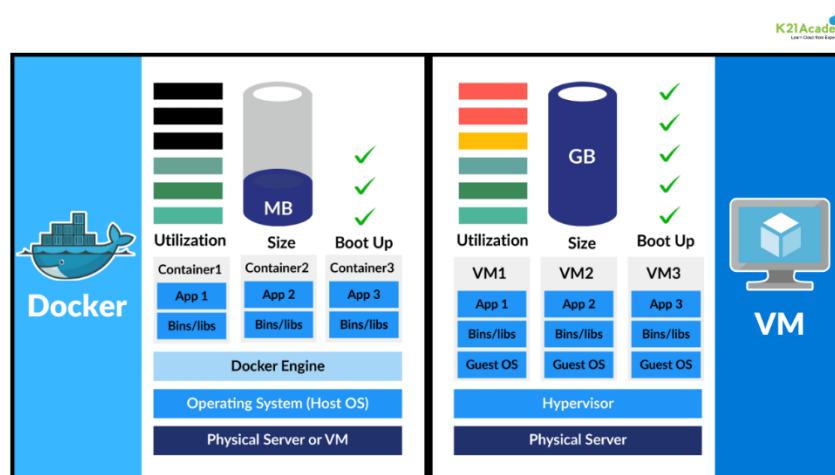
Selain model IaaS, perkembangan teknologi *cloud computing* juga menghadirkan konsep Container as a Service (CaaS) sebagai model layanan yang memanfaatkan teknologi *container* dalam proses penyediaan aplikasi. CaaS berada di antara model Infrastructure as a Service (IaaS) dan Platform as a Service (PaaS) serta memungkinkan pengguna untuk melakukan proses *deployment*, pengelolaan, dan orkestrasi *container* melalui platform seperti Kubernetes. Pendekatan ini menawarkan pemanfaatan sumber daya yang lebih efisien, kemudahan dalam melakukan skalabilitas, serta fleksibilitas yang lebih tinggi pada proses *deployment* aplikasi dibandingkan pendekatan berbasis *virtual machine* (Mahatme & Satpute, 2025; Şenel et al., 2023).

2.1.2 Virtualisasi dan Containerization

Virtualisasi merupakan teknologi yang memungkinkan satu perangkat keras fisik digunakan untuk menjalankan beberapa lingkungan komputasi secara independen. Teknologi ini banyak diterapkan pada infrastruktur *cloud* karena mampu meningkatkan pemanfaatan sumber daya melalui pembentukan mesin virtual (*Virtual Machine* atau VM). Setiap VM beroperasi secara terpisah dengan memiliki sistem operasi, aplikasi, serta perangkat lunak pendukungnya sendiri. Meskipun memberikan tingkat isolasi yang tinggi, pendekatan ini membutuhkan alokasi CPU, RAM, dan media penyimpanan yang relatif besar karena setiap mesin virtual harus menjalankan kernel sistem operasinya masing-masing (Bhardwaj et al., 2025).

Seiring berkembangnya kebutuhan akan infrastruktur yang lebih ringan dan efisien, teknologi *containerization* diperkenalkan sebagai alternatif terhadap virtualisasi berbasis VM. Berbeda dengan mesin virtual, container tidak memerlukan sistem operasi tersendiri, melainkan memanfaatkan kernel dari sistem operasi *host*. Setiap container hanya membawa aplikasi beserta seluruh dependensi yang dibutuhkan, seperti library, konfigurasi, dan *runtime*, sehingga proses distribusi maupun eksekusi aplikasi dapat dilakukan dengan lebih sederhana dan efisien (Bentaleb et al., 2022).

Pada penelitian ini, teknologi Docker Container digunakan sebagai implementasi konsep *containerization* untuk menggantikan pendekatan virtualisasi tradisional dalam pengelolaan layanan *cloud hosting*. Pemanfaatan Docker memungkinkan beberapa aplikasi web dijalankan secara terisolasi pada satu server fisik dengan penggunaan sumber daya yang lebih efisien. Pendekatan tersebut mendukung proses otomatisasi pengelolaan layanan *cloud hosting* sehingga setiap aplikasi dapat dikelola secara mandiri tanpa mengurangi performa maupun stabilitas sistem (Şenel et al., 2023).



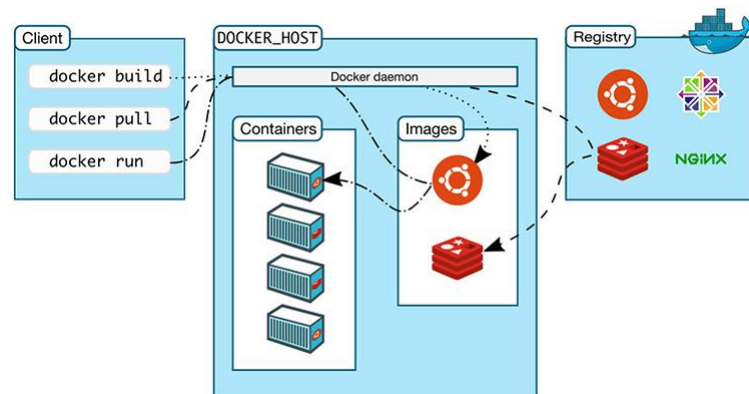
Gambar 2. 1 Perbandingan Virtual Machine (VM) vs Docker Container.

(Sumber: www.K2Academy.com, 2024)

2.1.3 Docker dan Docker Engine

Docker merupakan platform *open source* yang dikembangkan untuk mempermudah proses pembangunan, distribusi, dan eksekusi aplikasi menggunakan teknologi *container*. Dengan memanfaatkan *container*, aplikasi dapat dijalankan pada berbagai lingkungan komputasi secara konsisten karena seluruh kebutuhan aplikasi, seperti *library*, konfigurasi, dan *runtime*, dikemas dalam satu kesatuan. Pada penelitian ini, Docker digunakan sebagai fondasi utama dalam penyediaan layanan *cloud hosting*, di mana setiap aplikasi yang dideploy pengguna dijalankan di dalam *container* yang saling terisolasi sehingga tidak terjadi konflik dependensi antar aplikasi (Alyas et al., 2022).

Operasional Docker didukung oleh Docker Engine, yaitu komponen utama yang bertugas mengelola seluruh aktivitas *container*. Docker Engine menggunakan arsitektur *client-server* yang terdiri atas tiga komponen utama, yaitu Docker Daemon, REST API, dan Docker Client. Docker Daemon (*dockerd*) berjalan sebagai layanan di sistem operasi Linux dan bertanggung jawab mengelola berbagai objek Docker, seperti *image*, *container*, *network*, dan *volume*. Selanjutnya, REST API berfungsi sebagai media komunikasi antara aplikasi eksternal dengan Docker Daemon sehingga berbagai proses administrasi dapat dilakukan secara terprogram. Pada penelitian ini, backend yang dibangun memanfaatkan REST API untuk mengotomatisasi berbagai proses, seperti pembuatan, penghentian, maupun penghapusan *container*. Adapun Docker Client atau *Docker CLI* digunakan sebagai antarmuka berbasis *command line* untuk mengirimkan perintah kepada Docker Daemon ketika proses administrasi dilakukan secara manual.



Gambar 2. 2 Arsitektur Docker (Client - Docker Host/Daemon - Registry).

(Sumber: www.nickjanetakis.com, 2017)

Dalam implementasinya, Docker memiliki beberapa komponen penting yang mendukung proses pengelolaan *container*. Docker Image berfungsi sebagai *template* yang berisi seluruh kebutuhan aplikasi dan digunakan sebagai dasar dalam pembuatan *container*. Ketika *image* dijalankan, Docker akan membentuk sebuah Docker Container yang menjadi lingkungan eksekusi aplikasi secara terisolasi.

Proses pembuatan *image* dilakukan menggunakan Dockerfile, yaitu berkas konfigurasi yang berisi serangkaian instruksi pembangunan *image*.

Pemanfaatan Docker Engine pada penelitian ini memungkinkan proses penyediaan layanan *cloud hosting* dilakukan secara lebih efisien. Setiap pengguna dapat memiliki lingkungan aplikasi yang berjalan secara terisolasi pada satu server fisik tanpa mengganggu aplikasi milik pengguna lain. Pendekatan tersebut mendukung peningkatan efisiensi penggunaan sumber daya sekaligus mempermudah proses pengelolaan dan skalabilitas layanan (Alyas et al., 2022).

Selain Docker, terdapat *container runtime* lain yang cukup banyak digunakan, yaitu Podman. Beberapa penelitian menunjukkan bahwa Docker dan Podman memiliki tingkat performa yang relatif sebanding, namun masing-masing menawarkan karakteristik yang berbeda. Podman unggul pada aspek keamanan karena menggunakan arsitektur *daemonless*, sedangkan Docker lebih dikenal karena memiliki ekosistem yang matang, dokumentasi yang lengkap, serta kemudahan integrasi dengan berbagai platform pengembangan. Pada beberapa skenario pengujian dengan beban kerja tinggi, Podman mampu memberikan performa yang lebih baik, sedangkan Docker tetap menunjukkan kinerja yang optimal pada implementasi layanan *container* dengan kebutuhan yang beragam (Dordevic et al., 2022; Voulgaris et al., 2022).

2.1.4 Docker Compose

Docker Compose merupakan perangkat lunak yang digunakan untuk mengelola aplikasi yang terdiri atas beberapa container secara bersamaan. Melalui Docker Compose, seluruh layanan yang dibutuhkan oleh sebuah aplikasi dapat didefinisikan dalam satu berkas konfigurasi berformat YAML (*docker-compose.yml*), sehingga proses pembangunan dan pengoperasian lingkungan aplikasi menjadi lebih sederhana. Berbeda dengan Docker Engine yang mengelola container secara individual, Docker Compose memungkinkan beberapa layanan dijalankan dan dikonfigurasi sebagai satu kesatuan sistem (Eng et al., 2024; Ibrahim et al., 2021).

Pada penelitian ini, Docker Compose dimanfaatkan untuk mendukung proses pengembangan dan implementasi sistem berbasis NestJS dengan mengatur berbagai layanan yang saling bergantung. Melalui berkas *docker-compose.yml*, setiap layanan dapat dikonfigurasi, termasuk *image* yang digunakan, pemetaan *port*, variabel lingkungan (*environment variables*), jaringan (*network*), serta penyimpanan data (*volume*).

Penerapan *Docker Compose* memberikan beberapa manfaat pada sistem yang dikembangkan. Pertama, seluruh layanan, seperti aplikasi backend dan PostgreSQL, dapat dijalankan dalam jaringan yang sama sehingga komunikasi antar-*container* berlangsung secara lebih stabil dan terisolasi. Kedua, penggunaan fitur *volume* memungkinkan data PostgreSQL tetap tersimpan meskipun *container* dihentikan atau dibuat ulang, sehingga proses *data persistence* dapat dipertahankan.

Dengan memanfaatkan Docker Compose, lingkungan *cloud hosting* yang dibangun pada penelitian ini memiliki konfigurasi yang lebih terstruktur, mudah direplikasi, serta lebih sederhana dalam proses pemeliharaan dan pengembangan. Pendekatan tersebut juga mendukung konsistensi konfigurasi antara lingkungan pengembangan, pengujian, dan implementasi sehingga proses *deployment* aplikasi dapat dilakukan secara lebih efektif (Eng et al., 2024).

2.1.5 JavaScript Runtime: Node.js dan NestJS

Node.js merupakan lingkungan *runtime* JavaScript yang bersifat *open-source* dan dapat dijalankan pada berbagai sistem operasi. Platform ini dibangun menggunakan mesin Google V8 JavaScript Engine sehingga mampu mengeksekusi kode JavaScript di luar lingkungan *browser*. Salah satu karakteristik utama Node.js adalah penggunaan arsitektur *event-driven* dan mekanisme *non-blocking input/output (I/O)* yang memungkinkan aplikasi menangani banyak proses secara bersamaan dengan penggunaan sumber daya yang lebih efisien. Karakteristik tersebut menjadikan Node.js banyak dimanfaatkan dalam pengembangan aplikasi yang membutuhkan komunikasi data secara *real-time* maupun layanan backend dengan tingkat konkurensi yang tinggi.

Pada penelitian ini, Node.js digunakan sebagai lingkungan eksekusi pada sisi server untuk menjalankan seluruh logika backend yang berkaitan dengan proses otomatisasi layanan *cloud hosting*. Melalui Node.js, sistem dapat berkomunikasi dengan Docker SDK, mengakses sistem berkas (*file system*) pada server Linux, serta menangani berbagai permintaan (*request*) dari antarmuka pengguna secara efisien.

Untuk meningkatkan kualitas pengembangan backend, penelitian ini memanfaatkan NestJS, yaitu *framework* berbasis Node.js yang dibangun menggunakan TypeScript. NestJS mengadopsi pendekatan Object Oriented Programming (OOP), Functional Programming (FP), dan Functional Reactive Programming (FRP) sehingga mampu menghasilkan struktur aplikasi yang lebih terorganisasi, mudah dikembangkan, serta sesuai untuk membangun sistem berskala besar.

Kombinasi antara performa Node.js dan arsitektur modular yang ditawarkan NestJS menghasilkan backend yang mampu menjalankan proses otomatisasi infrastruktur secara stabil. Pendekatan tersebut mendukung pengelolaan berbagai layanan *cloud hosting*, mulai dari komunikasi dengan Docker Engine, pengelolaan basis data, hingga penyediaan layanan API yang digunakan oleh antarmuka pengguna secara terintegrasi. (Muhammad Zakia Avlach & Yamasari, 2025; Pratama & Raharja, 2023; Zima & Barszcz, 2024).

2.1.6 Database: PostgreSQL dan Prisma ORM

PostgreSQL merupakan *Relational Database Management System* (RDBMS) bersifat *open-source* yang banyak digunakan dalam pengembangan aplikasi karena memiliki kemampuan dalam menjaga konsistensi data, mendukung standar SQL, serta mampu menangani pengelolaan data dalam skala besar. Karakteristik tersebut menjadikan PostgreSQL sesuai untuk digunakan pada aplikasi yang membutuhkan keandalan, keamanan, dan performa basis data yang stabil. Dalam penelitian ini, PostgreSQL berfungsi sebagai media penyimpanan utama untuk mengelola

berbagai data operasional, seperti informasi pengguna, konfigurasi *container*, riwayat aktivitas sistem, serta status layanan *cloud hosting* yang sedang berjalan.

Untuk mempermudah proses interaksi antara aplikasi backend dengan PostgreSQL, penelitian ini memanfaatkan Prisma ORM (Object-Relational Mapping) sebagai lapisan abstraksi data. Prisma memungkinkan aplikasi yang dikembangkan menggunakan JavaScript maupun TypeScript berkomunikasi dengan basis data melalui antarmuka pemrograman yang lebih sederhana tanpa harus menuliskan perintah SQL secara langsung. Pendekatan ini membantu meningkatkan produktivitas pengembangan sekaligus mengurangi potensi kesalahan ketika mengelola data aplikasi (Salunke & Ouda, 2024; Yusmita et al., 2025; Zhadko-Bazilevych, 2025).

Penerapan Prisma ORM memberikan beberapa keuntungan dalam proses pengembangan sistem. Pertama, Prisma menyediakan mekanisme type safety melalui *Prisma Client* yang secara otomatis menghasilkan tipe data berdasarkan struktur basis data sehingga kesalahan penulisan *query* dapat diminimalkan selama proses pengembangan.

Kombinasi PostgreSQL sebagai sistem manajemen basis data dan Prisma ORM sebagai lapisan akses data memberikan fondasi yang kuat bagi sistem *cloud hosting* yang dikembangkan. Seluruh data operasional dapat dikelola secara lebih terstruktur, aman, dan konsisten, sementara proses komunikasi antara aplikasi backend dan basis data berlangsung secara efisien sehingga mampu mendukung kebutuhan sistem yang terus berkembang (Salunke & Ouda, 2024; Yusmita et al., 2025; Zhadko-Bazilevych, 2025).

2.1.7 Komunikasi Server: SSH2 Library

SSH2 Library merupakan pustaka (*library*) pada lingkungan Node.js yang menyediakan implementasi protokol Secure Shell (SSH2) menggunakan JavaScript. Protokol SSH2 dirancang untuk membangun komunikasi yang aman antara dua perangkat melalui jaringan dengan menyediakan mekanisme autentikasi, enkripsi data, eksekusi perintah jarak jauh (*remote command execution*), serta

transfer berkas secara aman. Pemanfaatan protokol ini banyak digunakan dalam administrasi server karena mampu menjaga kerahasiaan data dan mencegah akses yang tidak sah selama proses komunikasi berlangsung (Electronics and Communication Dept., R.V College of Engineering, Bangalore, India. et al., 2021).

Penerapan SSH2 Library memberikan beberapa keuntungan dalam implementasi sistem. Pertama, backend dapat mengeksekusi berbagai perintah Command Line Interface (CLI), seperti docker run, docker stop, maupun docker compose up, tanpa mengharuskan administrator masuk ke server secara langsung. Kedua, seluruh komunikasi antara aplikasi dan server berlangsung menggunakan mekanisme enkripsi SSH sehingga informasi autentikasi maupun data perintah terlindungi selama proses transmisi. Ketiga, SSH2 Library menyediakan dukungan terhadap fitur interactive shell dan stream, yang memungkinkan keluaran (*output*) terminal dikirim secara *real-time*. Kemampuan tersebut dimanfaatkan untuk mengintegrasikan layanan terminal berbasis web menggunakan Xterm.js, sehingga pengguna dapat berinteraksi dengan server melalui antarmuka browser. Selain itu, penggunaan SSH2 Library memungkinkan aplikasi memproses keluaran perintah, mendeteksi kesalahan (*error handling*), serta memberikan respons secara otomatis kepada pengguna berdasarkan hasil eksekusi yang diterima dari server. (Electronics and Communication Dept., R.V College of Engineering, Bangalore, India. et al., 2021).

2.1.8 Interactive Web Terminal (Xterm.js)

Xterm.js merupakan pustaka (*library*) berbasis TypeScript yang dirancang untuk menghadirkan emulasi terminal secara langsung di dalam *web browser*. Dengan memanfaatkan Xterm.js, pengguna dapat berinteraksi dengan lingkungan *command line interface* (CLI) melalui antarmuka web tanpa perlu menggunakan aplikasi terminal terpisah. Kemampuan tersebut menjadikan Xterm.js banyak dimanfaatkan pada aplikasi administrasi server, layanan *cloud*, maupun platform yang membutuhkan akses terminal berbasis web (Xterm.js Project, 2024).

Dalam penelitian ini, Xterm.js digunakan sebagai antarmuka terminal interaktif yang terhubung dengan server Linux melalui SSH2 Library pada sisi backend. Integrasi tersebut memungkinkan setiap perintah yang dikirimkan pengguna dari browser diteruskan ke server, sedangkan hasil eksekusi perintah ditampilkan kembali secara langsung pada antarmuka terminal. Dengan mekanisme tersebut, pengguna dapat melakukan administrasi layanan *cloud hosting* tanpa harus membuka aplikasi SSH eksternal.

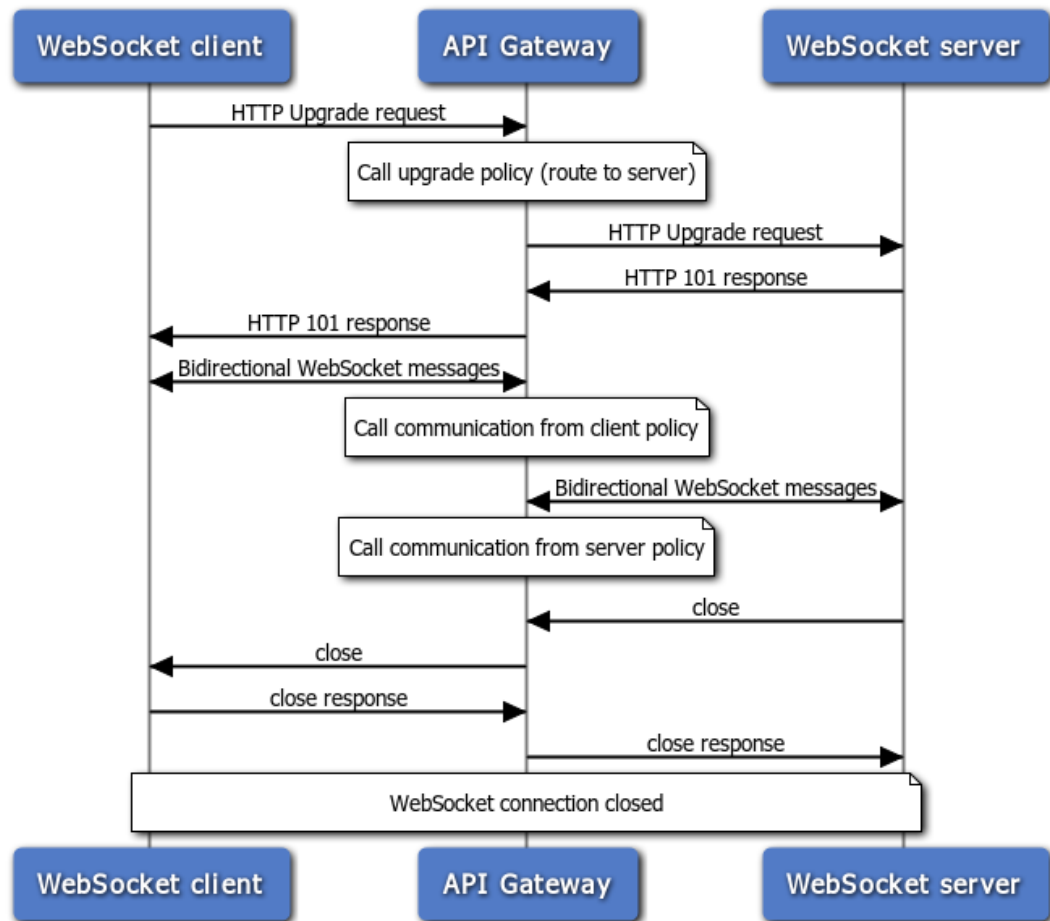
Penerapan Xterm.js memberikan berbagai manfaat terhadap pengelolaan sistem. Pertama, Xterm.js menyediakan fitur emulasi terminal yang mendukung berbagai kemampuan terminal modern, seperti pengaturan warna, dukungan interaksi menggunakan *mouse*, serta tampilan keluaran (*output*) yang menyerupai terminal asli sehingga pengalaman penggunaan menjadi lebih intuitif. Kedua, pengguna dapat menjalankan berbagai perintah administrasi secara langsung melalui browser tanpa melakukan konfigurasi SSH secara manual maupun memasang perangkat lunak terminal tambahan pada komputer lokal. Ketiga, melalui integrasi dengan SSH2 Library, Xterm.js mampu menampilkan keluaran perintah dari server secara *real-time* dengan latensi yang rendah. (Xterm.js Project, 2024; Deshpande et al., 2025; Murley et al., 2021)

2.1.9 Protokol Komunikasi: WebSocket

WebSocket merupakan protokol komunikasi yang memungkinkan pertukaran data secara dua arah (full-duplex) melalui satu koneksi *Transmission Control Protocol* (TCP) yang bersifat persisten. Berbeda dengan protokol HTTP yang menggunakan mekanisme *request-response* dan bersifat *stateless*, *WebSocket* memungkinkan server maupun *client* saling mengirimkan data kapan saja tanpa harus membangun koneksi baru untuk setiap proses komunikasi. Karakteristik tersebut menjadikan *WebSocket* sesuai untuk aplikasi yang membutuhkan pertukaran data secara *real-time*, seperti layanan *chat*, pemantauan sistem, maupun terminal berbasis web (Murley et al., 2021).

Pada penelitian ini, WebSocket dimanfaatkan sebagai media komunikasi antara antarmuka pengguna yang dibangun menggunakan Next.js dengan backend berbasis NestJS, khususnya pada fitur Interactive Web Terminal. Penggunaan WebSocket memungkinkan setiap aktivitas yang dilakukan pengguna pada terminal berbasis web dapat diteruskan ke server secara langsung, sementara hasil eksekusi perintah dikirim kembali tanpa memerlukan proses *refresh* halaman.

Mekanisme komunikasi WebSocket pada sistem diawali dengan proses handshake, yaitu ketika aplikasi *client* mengirimkan permintaan melalui protokol HTTP untuk meningkatkan koneksi menjadi WebSocket. Setelah proses tersebut berhasil, koneksi akan tetap terbuka (*persistent connection*) selama sesi terminal berlangsung sehingga komunikasi dapat dilakukan secara berkelanjutan. Setiap perintah yang dimasukkan pengguna melalui Xterm.js diteruskan ke backend menggunakan WebSocket, kemudian backend mengirimkan perintah tersebut ke server Linux melalui SSH2 Library. Selanjutnya, hasil eksekusi yang diterima dari server akan dikirim kembali ke browser secara *real-time* sehingga pengguna dapat melihat keluaran terminal secara langsung tanpa harus memuat ulang halaman. (Deshpande et al., 2025; Murley et al., 2021).



Gambar 2. 3 Diagram Websocket Connection Client dan Server

(Sumber: www.docs.oracle.com, 2016)

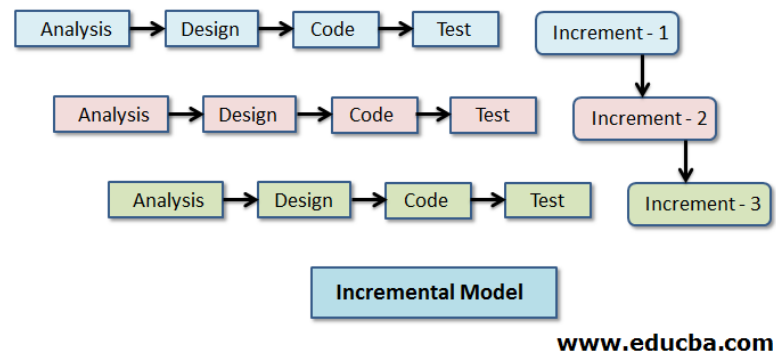
2.1.10 Metode Pengembangan Sistem: *Incremental Model*

Metode Incremental merupakan salah satu model pengembangan perangkat lunak yang membangun sistem secara bertahap melalui pembagian fitur ke dalam beberapa modul atau *increment*. Setiap *increment* menghasilkan bagian sistem yang telah berfungsi dan dapat diuji secara mandiri sebelum dilanjutkan ke tahap pengembangan berikutnya. Pendekatan ini memungkinkan proses pembangunan perangkat lunak berlangsung secara lebih terstruktur, sehingga setiap penambahan fitur dapat dilakukan tanpa harus menunggu seluruh sistem selesai dikembangkan (Shaheen et al., 2025; Sutjiadi et al., 2022).

Pemilihan metode Incremental pada penelitian ini didasarkan pada karakteristik sistem otomatisasi *cloud hosting* yang terdiri atas beberapa komponen dengan fungsi yang saling berkaitan, seperti pengelolaan *container*, administrasi server, autentikasi pengguna, serta layanan *Interactive Web Terminal*. Dengan membagi proses pengembangan ke dalam beberapa tahap, setiap modul dapat dirancang, diimplementasikan, dan divalidasi secara bertahap sehingga risiko kesalahan selama proses pengembangan dapat diminimalkan.

Penerapan metode Incremental memberikan beberapa manfaat dalam pelaksanaan penelitian. Pertama, proses pengembangan dapat difokuskan pada fitur yang menjadi fondasi sistem, seperti konfigurasi server, basis data, dan Docker, sebelum dilanjutkan pada fitur yang lebih kompleks, termasuk otomatisasi *deployment* serta layanan terminal berbasis web. Kedua, setiap modul yang telah selesai dapat langsung diuji secara independen sehingga proses verifikasi tidak perlu menunggu seluruh sistem selesai dikembangkan (Sutjiadi et al., 2022). Ketiga, apabila terjadi perubahan kebutuhan atau ditemukan kendala pada salah satu komponen, misalnya integrasi SSH2 Library, perbaikan dapat dilakukan pada modul terkait tanpa memengaruhi modul lain yang telah stabil. Selain itu, pendekatan bertahap ini memudahkan peneliti dalam memantau perkembangan implementasi karena setiap tahap menghasilkan peningkatan fungsi sistem yang dapat dievaluasi secara langsung (Badri & Sari, 2023; Sutjiadi et al., 2022).

Dalam penelitian ini, proses pengembangan diawali dengan pembangunan arsitektur backend beserta basis data sebagai fondasi sistem. Tahap berikutnya difokuskan pada integrasi Docker Engine dan layanan pendukung untuk mendukung otomatisasi pengelolaan *container*. Setelah seluruh layanan backend berjalan dengan baik, pengembangan dilanjutkan pada implementasi antarmuka pengguna berbasis Next.js, termasuk integrasi dashboard dan fitur *Interactive Web Terminal*, hingga seluruh modul membentuk sistem otomatisasi *cloud hosting* yang terintegrasi.



Gambar 2. 4 Diagram Metode Incremental

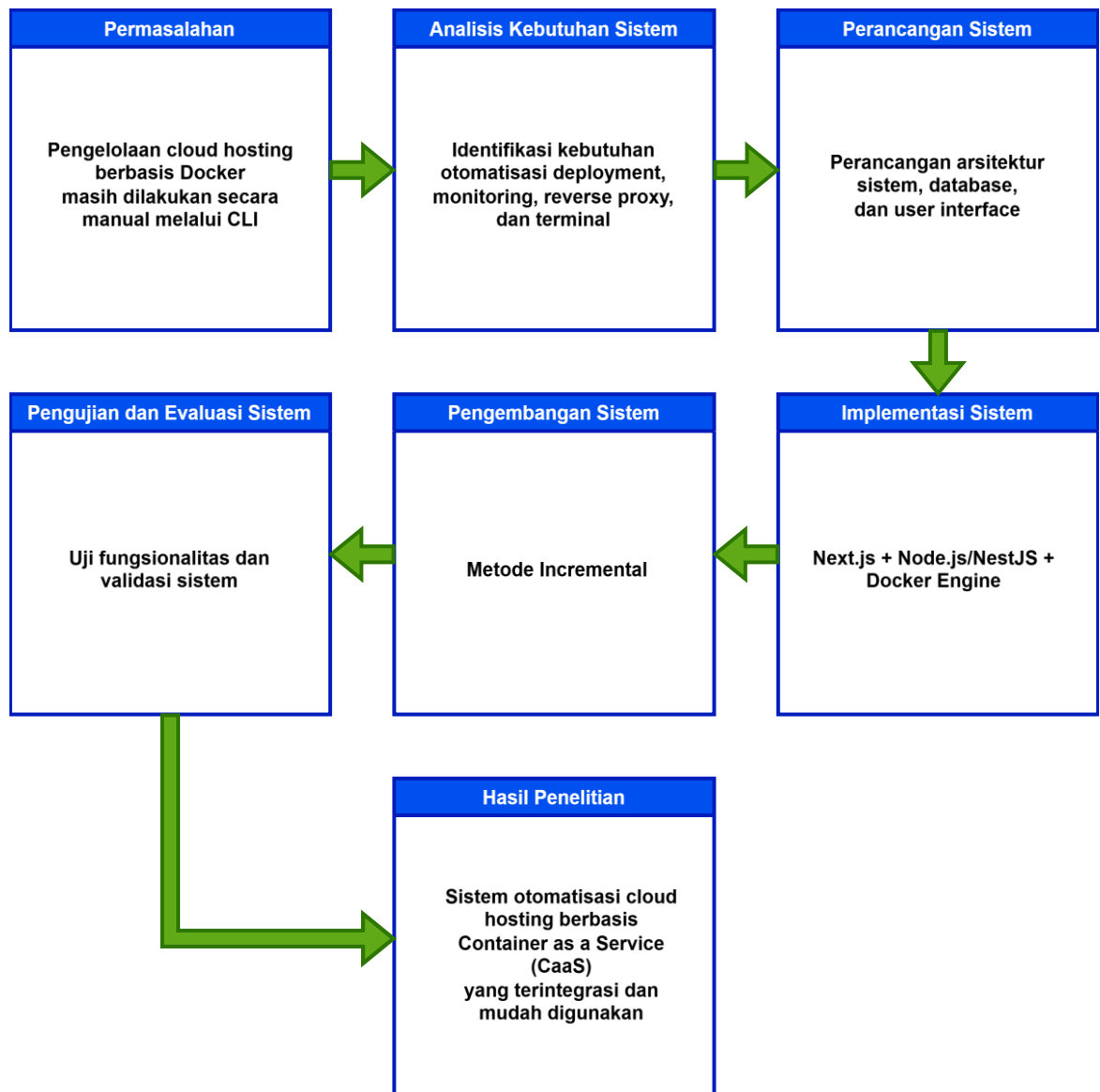
(Sumber: www.educba.com, 2020)

2.2 Penelitian Terdahulu

Penelitian terdahulu menjadi landasan utama dalam mengidentifikasi perkembangan teknologi serta celah penelitian (research gap) yang menjadi dasar pengembangan platform ini. Kajian literatur yang dilakukan mencakup tiga pilar utama, yaitu infrastruktur containerisasi berbasis Docker, kerangka kerja web modern (Next.js dan Node.js/NestJS), serta metodologi pengembangan sistem Incremental. Penelitian mengenai efektivitas dan otomatisasi Docker Container telah dilakukan oleh Ekaputra dan Affandi (2023), Hasan Amin dkk. (2025), Kiki Yuniar dkk. (2022), serta Nair et al. (2024) yang membuktikan bahwa containerization dan penggunaan reverse proxy mampu meningkatkan efisiensi dan performa deployment aplikasi. Namun, mayoritas penelitian tersebut masih berfokus pada analisis kinerja infrastruktur atau konfigurasi manual. Pada tingkat arsitektur yang lebih kompleks, Asif et al. (2025) dan Senel et al. (2023) mengeksplorasi konsep Container as a Service (CaaS) dan multi-tenant, tetapi belum mengimplementasikan antarmuka otomatisasi end-to-end yang terintegrasi secara visual untuk pengguna akhir. Lebih lanjut, validasi performa Docker dan keunggulan Node.js/NestJS sebagai controller infrastruktur didukung oleh hasil studi Pratama dan Raharja (2023) serta Husain et al. (2023). Dari sisi pengembangan antarmuka dan backend, Prasetyo dan Nugroho (2025) serta Munif dan Pelsri (2023) membuktikan efisiensi Next.js dalam menghasilkan web yang responsif, sementara Eli Nurhayati dan Agussalim (2023) menunjukkan keandalan Node.js dalam pengelolaan REST API. Guna memastikan seluruh modul platform ini dibangun secara terukur, penelitian ini mengadopsi metode Incremental sebagaimana yang diterapkan oleh Badri dan Fawaidul (2023), Riyan Ikhbal dkk. (2025), serta Halimah dkk. (2024). Perbedaan utama sekaligus kebaruan (novelty) dari penelitian yang diusulkan terletak pada integrasi platform Container as a Service (CaaS) bernama Portdock. Platform ini memadukan Next.js pada frontend dan Node.js/NestJS pada backend untuk mengotomatisasi penyediaan container Docker, konfigurasi reverse proxy otomatis, pemantauan sumber daya real-time, serta akses interactive web terminal berbasis Xterm.js dalam satu antarmuka terpusat.

2.3 Kerangka Pemikiran

Kerangka pemikiran dalam penelitian ini disusun sebagai landasan alur berpikir logis untuk memecahkan permasalahan pengelolaan infrastruktur cloud server. Alur kerangka pemikiran secara komprehensif disajikan pada gambar berikut



Gambar 2. 5 Diagram Kerangka Berfikir Penelitian

Berdasarkan Gambar 2.5 di atas, alur kerangka pemikiran penelitian ini dijelaskan sebagai berikut:

1. Permasalahan Penelitian diawali dari identifikasi masalah utama, yaitu pengelolaan dan penyediaan (provisioning) cloud hosting berbasis Docker yang saat ini masih dilakukan secara manual melalui Command Line Interface (CLI). Hal ini memerlukan kemampuan teknis tinggi dan berisiko menimbulkan kesalahan konfigurasi (human error).
2. Analisis Kebutuhan Sistem Berdasarkan permasalahan tersebut, dilakukan analisis kebutuhan untuk menentukan fitur-fitur utama platform, mencakup otomatisasi deployment container, pemantauan sumber daya (monitoring real-time), konfigurasi reverse proxy otomatis, serta penyediaan interactive web terminal.
3. Perancangan Sistem Tahap ini meliputi perancangan arsitektur sistem terintegrasi, perancangan skema basis data, serta perancangan antarmuka pengguna (User Interface) yang intuitif dan responsif.
4. Implementasi Sistem Sistem diimplementasikan menggunakan stack teknologi modern, yaitu Next.js pada sisi frontend, Node.js/NestJS pada sisi backend controller, serta Docker Engine sebagai lingkungan runtime container.
5. Pengembangan Sistem (Metode Incremental) Proses pengembangan platform menerapkan metode Incremental. Pendekatan ini memungkinkan setiap modul (seperti pembuatan container, reverse proxy, dan web terminal) dikembangkan, diuji, dan dievaluasi secara bertahap dan berulang untuk menjamin kualitas dan stabilitas sistem.
6. Pengujian dan Evaluasi Sistem Setelah modul dibangun, dilakukan uji fungsionalitas dan validasi sistem untuk memastikan seluruh fitur berjalan sesuai dengan spesifikasi kebutuhan yang telah ditetapkan.
7. Hasil Penelitian Hasil akhir dari penelitian ini adalah terciptanya platform Container as a Service (CaaS) untuk otomatisasi cloud hosting berbasis Docker yang terintegrasi, aman, dan mudah digunakan oleh pengguna tanpa ketergantungan pada perintah CLI manual.

2.4 Posisi Penelitian

Penelitian mengenai otomatisasi infrastruktur dan penggunaan teknologi kontainerisasi telah banyak dilakukan sebelumnya. Posisi penelitian ini bertujuan untuk memetakan keterkaitan serta perbedaan antara penelitian yang sedang dilakukan dengan penelitian-penelitian terdahulu, sehingga dapat menunjukkan perbedaan dan kontribusi dari sistem yang dibangun. Perbandingan posisi penelitian ini disajikan dalam Tabel 1 berikut :

Tabel 2. 1 Tabel Perbandingan Penelitian Sebelumnya (State of the Art)

NO	Peneliti (Tahun)	Judul Penelitian	Metode	Objek Penelitian	Perbandingan yang dijadikan alasan tinjauan penelitian
1.	Achsan Rizky Ekaputra, & Arif Saivul Affandi (2023)	Pemanfaatan layanan cloud computing dan docker container untuk meningkatkan kinerja aplikasi web Jurnal: <i>Journal of Information System and Application Development</i>	Eksperimental	menganalisis dan mengimplement asikan pemanfaatan layanan Content Delivery Network (CDN) dan cloud computing pada aplikasi web menggunakan Nginx Proxy Manager dan docker container	Hasil penelitian ini digunakan sebagai referensi mengenai efektivitas penggunaan Docker Container dan Nginx Proxy Manager dalam lingkungan <i>cloud</i> . Namun, penelitian tersebut hanya berfokus pada analisis kinerja infrastruktur, sedangkan pada penelitian ini dikembangkan sebuah User Interface (UI) untuk mengotomatisasi seluruh proses tersebut, sehingga memudahkan

		Vol/No/Hal: Vol. 1, Hal. 138-147 p-ISSN / e- ISSN: 2988- 4721, 2988- 5698 DOI/URL: https://jurnal.unmer.ac.id/index.php/jisad/article/view/11084			pengguna dalam pembuatan <i>cloud</i> <i>hosting</i> tanpa konfigurasi manual
2.	Hasan Amin dkk, (2025)	Otomatisasi Deployment dan Manajemen Multi- Kontainer Docker untuk Skalabilitas dan Efisiensi Lingkungan Komputasi Terisolasi	Eksperimental Teknis	merancang dan mengimplement asikan otomasi <i>deploy</i> <i>ment</i> aplikasi berbasis <i>Docker</i> guna meningkatkan efisiensi dan konsistensi pengelolaan layanan	Hasil penelitian ini memberikan patokan mengenai pentingnya otomatisasi <i>deployment</i> untuk menjaga konsistensi dan efisiensi pengelolaan layanan pada kontainer Docker. Perbedaan dengan penelitian yang saya lakukan terletak pada pengembangan antarmuka berbasis

		Jurnal: Jurnal Sistem Informasi, Teknik Informatika dan Teknologi Pendidikan Vol/No/Hal: Vol. 5, Hal. 119-120 p-ISSN / e-ISSN: 2828-7924 DOI/URL: [https://loddosinstitute.org/journal/index.php/JUSTIKPEN/article/view/261] (10.55338/justikpen.v5i1.261)			web (UI) menggunakan Next.js dan Node.js yang memungkinkan proses manajemen kontainer dilakukan secara visual, serta penambahan fitur <i>Web Terminal</i> untuk akses langsung ke dalam kontainer yang tidak dibahas dalam penelitian tersebut.
3.	Badri & Fawaidul (2023)	Sistem Informasi Manajemen Penjualan Obat Pada Apotek Berbasis Web Dengan	Incremental	Rancang bangun sistem informasi manajemen penjualan obat berbasis web dengan metod incremental	Penelitian oleh Badri & Fawaidul (2023) dijadikan referensi penggunaan metode Incremental dalam pengembangan sistem informasi berbasis

		Menggunakan Metode Incremental Model Jurnal: Jurnal Sistem Informasi Dan Informatika Vol/No/Hal: Vol. 1, Hal. 70-79 p-ISSN / e-ISSN: 2988-4721, 2988-5698 DOI/URL: [https://jurnal.unidha.ac.id/index.php/jiska/article/view/960]			web. Perbedaannya terletak pada domain objek dan kompleksitas sistem: penelitian rujukan berfokus pada manajemen inventaris dan transaksi penjualan obat (Apotek), sedangkan penelitian yang dilakukan berfokus pada otomatisasi infrastruktur <i>cloud hosting</i> berbasis Docker (CaaS) untuk penyediaan <i>container</i>, <i>reverse proxy</i>, dan <i>web terminal</i>.
4.	Riyan Ikhbal Salam dkk, (2025)	Implementasi Incremental Method Pada Rancang Bangun Website	Incremental	merancang dan mengimplementasikan sebuah website yang dapat	Hasil penelitian ini digunakan sebagai acuan dalam mengimplementasikan tahapan metode <i>incremental</i> pada

		Penerbit Pnp Press Jurnal: Jurnal Teknoif Teknik Informatika Institut Teknologi Padang Vol/No/Hal: Vol. 13, Hal. 1-7 p-ISSN / e-ISSN: 2598-9197, 2338-2724 DOI/URL: [https://teknoif.itp.ac.id/index.php/teknoif/article/view/841] (10.21063/jtif.2025.V13.1.1-7)		memfasilitasi proses publikasi dan distribusi majalah dan buku di PNP Press menggunakan metode Incremental	pengembangan sistem berbasis web. Perbedaan utama dengan penelitian yang saya lakukan terletak pada objek dan kompleksitas sistem, di mana penelitian ini berfokus pada publikasi konten, sedangkan penelitian saya menerapkan metode tersebut untuk membangun sistem otomatisasi <i>cloud hosting</i> yang melibatkan manajemen <i>container Docker</i> dan integrasi <i>web terminal</i>.
5.	Subekti & Zaenal Mutaqin	Rancang Bangun Cloud Storage	PPDIOO (Prepare, Plan, Design,	merancang, mengimplementasikan serta	Hasil penelitian ini memberikan referensi mengenai

	(2025)	Berbasis Docker Container Jurnal: Jurnal PASTI: Jurnal Publikasi Artikel Sistem Teknologi Informasi Vol/No/Hal: Vol. 1, Hal. 30-39 p-ISSN / e-ISSN: DOI/URL: [https://ejournal.utmj.ac.id/jurnalpasti/article/view/968](10.9001/jurnalpasti.v1i1.968)	Implement, Operate, Optimize).	menguji kelayakan sistem cloud storage berbasis Docker container menggunakan layanan ownCloud sebagai solusi penyimpanan data di sekolah.	implementasi Docker Container untuk membangun layanan <i>Cloud Storage</i> yang stabil menggunakan metode PPDIOO. Perbedaannya dengan penelitian yang saya lakukan terletak pada fokus layanan dan metode pengembangan, di mana penelitian saya berfokus pada otomatisasi pembuatan cloud hosting untuk aplikasi web dan menggunakan Metode Incremental yang menekankan pada pengembangan fitur sistem secara bertahap.
6.	Halimah dkk, (2024)	Penerapan Model Incremental Pada Implementasi Dan Pengujian Sistem Informasi	Incremental	mengimplementasikan Sistem Informasi Pendanaan Usaha Mikro Kecil (PUMK) menggunakan metode	Hasil penelitian ini digunakan sebagai referensi mengenai keberhasilan implementasi metode <i>incremental</i> dalam meningkatkan efisiensi pengelolaan data pada

		Pendanaan Usaha Mikro Kecil (Pumk): Application Of The Incremental Model In The Implementation And Testing Of Small Micro Enterprises Funding Information Systems Jurnal: Jurnal Sistem Informasi dan Bisnis Cerdas Vol/No/Hal: Vol. 17, Hal. 11-20 p-ISSN / e-ISSN: 2686-0775, 2087-0639 DOI/URL: [https://sibc.unjati.ac.id/in]	incremental dalam upaya meningkatkan efisiensi pengelolaan usaha mikro kecil (UMK) di PT. Angkasa Pura II Kantor Cabang Jambi.	sistem informasi berbasis web. Perbedaannya terletak pada ranah aplikasi yang dibangun, di mana penelitian tersebut berfokus pada manajemen administrasi pendanaan usaha, sedangkan penelitian saya menerapkan metode ini untuk membangun platform otomatisasi infrastruktur <i>cloud</i> yang mengintegrasikan Docker, Node.js, dan fitur <i>interactive terminal</i>.
--	--	---	---	--

		dex.php/sibc/article/view/284 l(10.33005/sibc.v17i2.284)			
7.	Munif Sanjaya, & Pelsri Ramadar Noor Saputra (2023)	Pemanfaatan Nextjs Dan Mongodb Dalam Sistem Informasi Web Manajemen Data Beras Pada Ud Sri Utami Jurnal: INFORMATI ON SYSTEM FOR EDUCATORS AND PROFESSION ALS : Journal of Information System Vol/No/Hal: Vol. 8, Hal. 25 p-ISSN / e-ISSN: 2548-3587	Waterfall	Pemanfaatan NextJS karena memiliki konsep Server-Side Rendering dan MongoDB berbasis dokumen (JSON) berbasis Cloud	Hasil penelitian ini memberikan validasi mengenai efektivitas penggunaan Next.js dan konsep <i>Server-Side Rendering</i> (SSR) untuk membangun sistem informasi berbasis <i>cloud</i> yang responsif. Perbedaan dengan penelitian saya terletak pada metode pengembangan dan fungsi sistem, di mana penelitian tersebut menggunakan metode Waterfall untuk manajemen data komoditas, sedangkan penelitian saya menggunakan Metode Incremental untuk membangun platform otomatisasi infrastruktur hosting yang terintegrasi dengan Docker.

		DOI/URL: [http://ejournal-binainsani.ac.id/index.php/ISBI/article/view/2414](10.51211/isbi.v8i1.2414)			
8.	Rakhmat Prasetyo & Agung Nugroho (2025)	Meningkatkan Performa Frontend dengan Menggunakan Framework Next.Js dalam Pengembangan Website Jurnal: Journal of Cyber Health and Computer Vol/No/Hal: Vol. 2, Hal. 14-19 p-ISSN / e-ISSN: 3031-4623	eksperimental komparatif	mengeksplorasi dampak penggunaan framework Next.js terhadap performa frontend website. Next.js, dengan fitur-fitur seperti server-side rendering (SSR) dan static site generation (SSG), menawarkan potensi peningkatan signifikan dalam kecepatan pemuatan halaman dan	Hasil penelitian ini memberikan bukti empiris mengenai keunggulan Next.js dalam meningkatkan kecepatan pemuatan halaman dan pengalaman pengguna melalui fitur SSR dan SSG. Perbandingan dengan penelitian saya adalah jurnal tersebut berfokus pada analisis performa <i>frontend</i> secara umum, sedangkan penelitian saya mengimplementasikan performa Next.js tersebut untuk membangun antarmuka manajemen infrastruktur yang

		DOI/URL: [https://www.jurnalsibermu.com/index.php/jochac/article/view/31] (10.64163/jochac.v2i2.31)		pengalaman pengguna.	responsif, yang terhubung langsung dengan <i>backend</i> Node.js untuk mengelola kontainer Docker secara <i>real-time</i> .
9.	Eli Nurhayati, & Agussalim (2023)	Rancang Bangun Back-end API pada Aplikasi Mobile AyamHub Menggunakan Framework Node JS Express Jurnal: Jurnal Sistem dan Teknologi Informasi (JustIN) Vol/No/Hal: Vol. 11, Hal. 524 p-ISSN / e-ISSN: 2620-8989, 2460-3562	Waterfall	perancangan arsitektur back-end untuk aplikasi mobile AyamHub menggunakan arsitektur REST dan framework Node.js Express.	Hasil penelitian ini memberikan referensi mengenai keunggulan Node.js dalam membangun arsitektur <i>back-end</i> yang efisien untuk aplikasi berbasis REST API. Perbedaan dengan penelitian saya terletak pada metode pengembangan dan tujuan sistem, di mana penelitian tersebut menggunakan metode Waterfall untuk aplikasi mobile, sedangkan penelitian saya menggunakan Metode Incremental untuk membangun sistem otomatisasi infrastruktur <i>cloud</i> yang mampu mengelola kontainer Docker secara dinamis

		DOI/URL: [https://jurnal.untan.ac.id/index.php/justin/article/view/66823] (10.26418/justin.v11i3.66823)			melalui <i>web antarmuka</i> .
10.	Kiki Yuniar & Kristiawan dkk, (2022)	Implementasi Reverse Proxy Pada Hosting Web Server Di Docker Container Jurnal: Jurnal Informatika dan Sistem Informasi Vol/No/Hal: Vol. 3, Hal. 1-12 p-ISSN / e-ISSN: 2722-130 DOI/URL: [http://jifosi.unjati.ac.id/index.php/jifosi/]	Eksperimental Teknis	melakukan analisa mendalam mengenai opini dalam penggunaan hosting pada container dengan menggunakan Docker container dan juga penerapan pada Reverse Proxy.	Hasil penelitian ini memberikan referensi teknis mengenai penggunaan Reverse Proxy untuk mengamankan <i>web server</i> di dalam Docker Container. Perbandingan dengan penelitian saya adalah jurnal tersebut berfokus pada analisis opini dan konfigurasi manual, sedangkan penelitian saya melakukan otomatisasi konfigurasi Reverse Proxy tersebut ke dalam sebuah platform manajemen hosting yang terpusat.

		article/view/422](10.33005/jifosi.v3i1.422)			
11.	Abhishek M Nair et al. (2024)	Dockerized Application with Web Interface Jurnal: International Journal of Scientific Research in Computer Science, Engineering and Information Technology Vol/No/Hal: Vol. 10, Hal. 412-419 p-ISSN / e-ISSN: 2456-3307 DOI/URL: [https://ijsrcseit.com/home/article/view/CSEIT243646](	Eksperimental dan Rancang Bangun	Otomatisasi deployment Docker melalui antarmuka web	Penelitian ini menjadi acuan utama karena memiliki kesamaan dalam membangun dashboard web untuk mengelola Docker. Perbedaanannya, penelitian tersebut menggunakan Flask API sebagai jembatan backend, sedangkan penelitian saya fokus pada integrasi Node.js/NestJS

		10.32628/CSE IT243646)			
12.	Asif et al. (2025)	Development of Container as a Service Management Platform and Architecture Utilizing Linux Container Jurnal: The 10th International Conference on Science and Technology Vol/No/Hal: Vol. 27, Hal. 23-30 p-ISSN / e-ISSN: 2813-8333 DOI/URL: [https://www.scientific.net/EH.27.23](10.4028/p-G4x83Y)	Perancangan arsitektur dan implementasi platform CaaS berbasis Linux Container serta pengujian sistem	Platform CaaS berbasis LXC dan Proxmox	Fokus pada pengembangan arsitektur dan platform CaaS berbasis Linux Container, namun belum membahas aspek performa dan otomatisasi infrastruktur secara menyeluruh.

13.	I Putu Agus Eka Pratama dan I Made Sunia Raharja (2023)	Node.js Performance Benchmarking and Analysis at Virtualbox, Docker, and Podman Environment Using Node-Bench Method Jurnal: JOIV : International Journal on Informatics Visualization Vol/No/Hal: Vol. 7, Hal. 22-40 p-ISSN / e-ISSN: 2549-9904, 2549-9610 DOI/URL: [http://joiv.org/index.php/joiv/article/view/1762](10.30630/joiv.7.4.01762)	Node-Bench (Eksperimental)	Analisis performa Node.js pada lingkungan Docker dan Virtualisasi	Penelitian ini memberikan validasi ilmiah bahwa Docker memiliki performa lebih unggul dibandingkan Virtual Machine dalam menjalankan aplikasi Node.js. Perbedaannya, penelitian tersebut fokus pada <i>benchmarking</i> berbagai <i>framework</i>, sedangkan penelitian saya berfokus pada implementasi praktis Node.js untuk mengontrol Docker Engine secara otomatis.
-----	---	--	-----------------------------	---	---

14.	Husain et al. (2023)	Analysis and Implementation of Comparison Between Podman and Docker in Container Management Jurnal: International Journal of Electronics and Communications Systems Vol/No/Hal: Vol. 3, Hal. 57-67 p-ISSN / e-ISSN: 2798-2610 DOI/URL: [https://ejournal.radenintan.ac.id/index.php/IJECS/article/view/19860](	Network Development Life Cycle (NDLC)	Perbandingan efisiensi manajemen kontainer pada Docker dan Podman	Penelitian ini memberikan referensi mengenai pengelolaan <i>registry</i> dan <i>user access</i> pada lingkungan kontainer. Perbedaannya, penelitian tersebut fokus pada perbandingan performa antar-platform, sedangkan penelitian saya fokus pada implementasi otomatisasi penyediaan layanan hosting menggunakan Docker Engine.
-----	-------------------------	--	---------------------------------------	---	--

		10.24042/ijecs.v3i2.19860)			
15.	Senel et al. (2023)	Multitenant Containers as a Service (CaaS) for Clouds and Edge Clouds Jurnal: IEEE Access Vol/No/Hal: Vol. 11, Hal. 144574-144601 p-ISSN / e-ISSN: 2169-3536 DOI/URL: https://ieeexplore.ieee.org/document/10365152/(10.1109/ACCESS.2023.3344486)	Perancangan framework multi-tenant CaaS serta implementasi dan evaluasi pada cloud dan edge cloud	Sistem CaaS multi-tenant berbasis Kubernetes	Menawarkan konsep multi-tenant CaaS untuk efisiensi resource pada cloud dan edge cloud, namun belum mengimplementasikan sistem otomatisasi berbasis Docker secara end-to-end.