LAPORAN TESIS

PENGUJIAN EFISIEN DAN PERFORMA KECEPATAN PADA APLIKASI *MICROSERVICE* WEB *SERVER* DENGAN *KUBERNETES* k0s, k3s, k3d, DAN k8s.

Oleh:

Fuad Achmadi

NIM: 22206051006

PROGRAM STUDI INFORMATIKA PROGRAM MAGISTER FAKULTAS SAINS DAN TEKNOLOGI UIN SUNAN KALIJAGA

YOGYAKARTA

2025

KEMENTERIAN AGAMA
UNIVERSITAS ISLAM NEGERI SUNAN KALIJAGA
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Marsda Adisucipto Telp. (0274) 540971 Fax. (0274) 519739 Yogyakarta 55281

PENGESAHAN TUGAS AKHIR

Nomor : B-1230/Un.02/DST/PP.00.9/06/2025

Tugas Akhir dengan judul : Pengujian Efisien Dan Performa Kecepatan Pada Aplikasi Microservice Web Server Dengan Kubernetes k0s, k3s, k3d, dan k8s.

yang dipersiapkan dan disusun oleh:

Nama : FUAD ACHMADI, A.m.k, S.Kom, M.E
Nomor Induk Mahasiswa : 22206051006
Telah diujikan pada : Rabu, 04 Juni 2025
Nilai ujian Tugas Akhir : A/B

dinyatakan telah diterima oleh Fakultas Sains dan Teknologi UIN Sunan Kalijaga Yogyakarta

TIM UJIAN TUGAS AKHIR

Ketua Sidang

Ir. Muhammad Taufiq Nuruzzaman, S.T. M.Eng., Ph.D.
SIGNED

Valid ID: 685b6822681a7

Penguji I

Dr. Ir. Bambang Sugiantoro, M.T., IPU., ASEAN Eng.
SIGNED

Valid ID: 684f90464e6a1

Penguji II

Dr. Ir. Aulia Faqih Rifa'i, M.Kom.
SIGNED

Valid ID: 685b665d24308

Yogyakarta, 04 Juni 2025
UIN Sunan Kalijaga
Dekan Fakultas Sains dan Teknologi

Prof. Dr. Dra. Hj. Khurul Wardati, M.Si.
SIGNED

Valid ID: 685cbd9a00cca

PERNYATAAN KEASLIAN

Yang bertanda tanga dibawah ini:

Nama : Fuad Achmadi
NIM : 22206051006
Jenjang : Magister
Program Studi : Informatika

Menyatakan bahwa naskah tesis ini yang secara keseluruhan adalah hasil penelitian/karya saya sendiri, kecuali pada bagian-bagian yang dirujuk sebelumnya.

Yogyakarta, 25 mei 2025
saya menyatakan,

Fuad Achmadi
NIM: 22206051006

KEMENTERIAN AGAMA REPUBLIK INDONESIA
UNIVERSITAS ISLAM NEGERI SUNAN KALIJAGA
MAGISTER INFORMATIKA
JI. Marsda Adisucipto Yogyakarta. Telp (0274) 515856 Yogyakarta 55281

SURAT PERSETUJUAN TUGAS AKHIR

Hal : Persetujuan Tugas Akhir
Kepada:
Yth. Dekan Fakultas Sains dan Teknologi

Uin Sunan Kalijaga Yogyakarta
Di Yogyakarta

Assalamu'alaikum Wr.Wb.
PENGUJIAN EFISIEN DAN PERFORMA KECEPATAN PADA APLIKASI MICROSERVICE WEB SERVER DENGAN KUBERNETES k0s, k3s, k3d, DAN k8s yang ditulis oleh:

Nama	: Fuad Achmadi
NIM	: 22206051006
Jenjang	: Magister
Program Studi	: Informatika

Sudah dapat diajukan kepada Program Studi Magister Informatika Fakultas Sains dan Teknologi UIN Sunan Kalijaga sebagai salah satu syarat untuk memperoleh gelar Magister
Informatika.
Dengan ini saya mengharap agar tugas tersebut di atas agar dapat segera di munaqosyahkan.
Atas perhatiannya saya ucapkan terimakasih.
Wassalamu'‘alaikum Wr. Wb.

Yogyakarta, 25 mei 2025
Pembimbing,

Ir. Muh. Taufiq Nuruzzaman, S.T. M.Eng., Ph.D.
19791118 200501 1 003

ABSTRAK

Kubernetes telah menjadi salah satu *platform* orkestrasi *container* yang paling populer untuk menjalankan aplikasi *microservice* di lingkungan *web server*. Namun, dengan berbagai *distribusi Kubernetes* seperti k8s, k3s, k3d, dan k0s, penting untuk memahami perbedaan efisiensi dan performa kecepatan masing-masing dalam berbagai skenario pengujian. Penelitian ini bertujuan untuk membandingkan efisiensi resource dan performa kecepatan dari keempat *platform* tersebut melalui *Parameter* seperti *throughput (RPS), latency* rata-rata, dan *network latency* internal. Hasil pengujian menunjukkan bahwa k3s menawarkan performa terbaik dengan *throughput* 2000–8000 RPS, *latency* 10–50 ms, dan *network latency internal* 0,5–2 ms, menjadikannya ideal untuk lingkungan dengan keterbatasan resource namun tetap membutuhkan performa tinggi. Sementara itu, k3d dan k0s menunjukkan performa yang lebih rendah dibandingkan k3s, dengan *throughput* masing-masing 1500–6000 RPS dan 1500–7000 RPS, serta *latency* 15–60 ms dan 20–70 ms, yang membuatnya cocok untuk pengembangan lokal atau produksi skala kecil hingga menengah. Sebaliknya, k8s memiliki *throughput* 1000–5000 RPS dengan *latency* tertinggi (50–200 ms) dan *network latency* internal 1–5 ms, menunjukkan bahwa *platform* ini lebih cocok untuk *workload* besar yang memerlukan *Skalabilitas* maksimal meskipun dengan *overhead* resource yang lebih tinggi. Kesimpulannya, pemilihan *platform Kubernetes* harus disesuaikan dengan kebutuhan spesifik, baik untuk pengembangan lokal, produksi skala kecil, maupun workload besar. k3s adalah solusi terbaik untuk performa tinggi dan efisiensi, sementara k8s tetap menjadi pilihan utama untuk fleksibilitas dan *Skalabilitas* maksimal.

Kata Kunci : efisiensi, performa, *microservice*, *Kubernetes*.

ABSTRACT

Kubernetes has become one of the most popular container orchestration platforms for running microservice applications in web server environments. However, with various Kubernetes distributions such as k8s, k3s, k3d, and k0s, it's important to understand the differences in efficiency and performance across multiple test scenarios. This study compares the resource efficiency and speed performance of the four platforms using Parameters such as throughput (RPS), average latency, and internal network latency. Test results show that k3s offers the best performance, achieving a throughput of 2000-8000 RPS, a latency of 10-50 ms, and internal network latency of 0.5-2 ms, making it ideal for resource-constrained environments requiring high performance. Meanwhile, k3d and k0s demonstrate lower performance than k3s, with throughputs of 1500-6000 RPS and 1500-7000 RPS, and latencies of 15-60 ms and 20-70 ms, respectively, making them suitable for local development or small- to medium-scale production. In contrast, k8s has a throughput of 1000-5000 RPS, the highest latency (50-200 ms), and internal network latency of 1-5 ms, indicating that this platform is better suited for large workloads that require maximum scalability, despite higher resource overhead. In conclusion, Kubernetes platform selection should be tailored to specific needs, whether for local development, small-scale production, or large workloads. K3s is the best solution for high performance and efficiency, while k8s remains the top choice for maximum flexibility and scalability.

Keywords: *Zero Trust Architecture, REST, Security, Microservices, Cyber Attack, API, JSON.*

MOTTO

Demi masa, sungguh, manusia berada dalam kerugian, kecuali orang-orang yang beriman dan mengerjakan kebajikan serta saling menasihati untuk kebenaran dan saling menasihati untuk kesabaran.
[103. QS. al-'Asr]

"Perumpamaan orang-orang yang menjadikan selain Allah sebagai pelindung adalah seperti laba-laba betina yang membuat rumah. Sesungguhnya rumah yang paling lemah ialah rumah laba-laba. Jika mereka tahu, (niscaya tidak akan menyembahnya)."

[*al-'Ankabut*:41]

"Kehidupan yang paling baik adalah kehidupan yang dihabiskan untuk memahami kebenaran"

[Ibnu Bajjah]

KATA PENGANTAR

Bismillahirahmanirrahim,

Assalamualaikum warahmatullahi wabarakatuhu

Puji syukur penulis panjatkan ke hadirat Allah SWT yang telah memberikan segala keindahan dan petunjuk-Nya, sehingga penulis dapat menyelesaikan tesis ini dengan judul "Pengujian Efisien Dan Performa Kecepatan Pada Aplikasi *Microservice Web Server* Dengan *Kubernetes* k0s, k3s, k3d, dan k8s." sebagai salah satu persyaratan mencapai gelar Magister Informatika.

Penulis ingin mengucapkan terima kasih kepada semua orang yang telah memberikan penghiburan, dukungan, dan pemikiran kreatif sehingga perencanaan proposal ini dapat diselesaikan sedikit demi sedikit. Terima kasih yang luar biasa diberikan kepada:

1. Bapak Prof. Dr. Phil. Al Makin, S.Ag., M.A., selaku Rektor UIN Sunan Kalijaga Yogyakarta.
2. Ibu Dr. Dra. Hj. Khurul Wardati, M.Si., selaku Dekan Fakultas Sains dan Teknologi UIN Sunan Kalijaga Yogyakarta.
3. Bapak Dr. Bambang Sugiantoro, M.T., selaku Ketua Program Studi Magister Informatika Fakultas Sains dan Teknologi UIN Sunan Kalijaga Yogyakarta.
4. Bapak Ir. Muhammad Taufiq Nuruzzaman, S.T. M.Eng., Ph.D. selaku dosen Pembimbing Tesis yang telah memberikan motivasi dan pengarahan selama studi sehingga dapat menyelesaikan penyusunan tesis ini.
5. Bapak dan Ibu dosen Program Studi Magister Informatika Fakultas Sains dan Teknologi UIN Sunan Kalijaga Yogyakarta yang sudah membagi ilmu yang sangat bermanfaat.
6. Seluruh Staf Karyawan Fakultas Sains dan Teknologi Universitas Islam Negeri Sunan Kalijaga Yogyakarta yang telah membantu sehingga penyusunan *thesis* ini

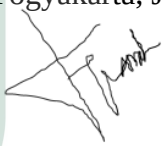
berjalan lancar.

7. Istri saya, Anis Fikriya, serta anak-anak saya, Tsabita asyfa dan Zea Almahyra Hafizhah yang senantiasa memberikan doa serta dukungan selama studi di Magister Informatika Universitas Islam Negeri Sunan Kalijaga Yogyakarta.
8. Teman-teman Magister Informatika angkatan 2022 Universitas Islam Negeri Sunan Kalijaga atas kerjasama, saran, dan bantuannya.
9. Semua pihak yang tidak bisa penulis sebutkan satu persatu atas bantuannya dalam menyelesaikan tesis ini.

Akhir kata, penulis hanya bisa bersyukur kepada Allah SWT, berharap apa yang telah dilakukan selama ini bisa menjadi amal dan persembahan berikut. Penulis menyadari bahwa dalam proses penulisan skripsi ini masih terdapat banyak kesalahan dan kekurangan, dan sangat mengharapkan kritik dan saran untuk perbaikan. Semoga skripsi ini dapat bermanfaat bagi penulis khususnya dan pembaca pada umumnya, terima kasih.

Wassalamualaikum warahmatullahi wabarakatuh

Yogyakarta, Juni 2025



Fuad Achmadi
22206051006

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR SINGKATAN

API	: *Application Programming Interface*
HTTP	: *HyperText Transfer Protocol*
Devops	: *Development and operation*
JSON	: *JavaScript Object Notation*
LDAP	: *Lightweight Directory Access Protocol*
OOP	: *Object-Oriented Programming*
RBAC	: *Role Based Access Control*
REST	: *REpresentational State Transfer*
SAML	: *Security Assertion Markup Language*
SQL	: *Structure Query Language*
URI	: *Uniform Resource Identifier*
XML	: *Extensible Markup Language*

BAB I

PENDAHULUAN

A. Latar Belakang

Pada era transformasi digital saat ini, banyak organisasi yang beralih dari *arsitektur monolitik* ke *arsitektur microservices* untuk mendukung inovasi dan efisiensi operasional. *Microservices* adalah gaya *arsitektur* di mana komponen-komponen dari suatu sistem dirancang sebagai layanan yang dapat di deploy secara *independen*. *Microservices* dirancang berdasarkan subdomain bisnis yang terdefinisi dengan baik, dan mereka berkomunikasi satu sama lain menggunakan *protokol* ringan, seperti HTTP (Peralta, 2023). *Microservices* menawarkan *fleksibilita*s dan *Skalabilitas* dengan memecah aplikasi besar menjadi layanan-*microservice* yang *independen*. Layanan-layanan ini berkomunikasi satu sama lain melalui REST API (*REpresentational State Transfer Application Programming Interface*), yang memungkinkan pertukaran data secara cepat dan efisien. REST merupakan jenis *web* service yang menerapkan konsep perpindahan antar *state*. *State* dapat digambarkan seperti jika *browser* meminta suatu halaman web, maka *server* akan mengirimkan *state* halaman web yang diminta ke *browser*. (Yusril, 2023). Sedangkan API (*Application Programming Interface*) adalah konsep krusial dalam dunia pengembangan perangkat lunak yang memungkinkan berbagai program atau aplikasi berkomunikasi dan berinteraksi satu sama lain. (Wiguna, 2023). Dalam praktiknya, pertukaran data pada REST API umumnya menggunakan data dalam bentuk JSON (*JavaScript Object Notation*) karena ringkas, lebih fleksibel dalam pertukaran format data, mudah dibaca, proses serialisasi dan deserialisasi yang cepat, serta didukung oleh berbagai macam bahasa pemrograman.

Namun, di balik keunggulan yang ditawarkan oleh *arsitektur microservices*, muncul tantangan keamanan yang semakin kompleks. *Arsitektur* ini memperbanyak titik-titik komunikasi yang rentan terhadap serangan siber, seperti MiTM (*man-in-the-middle*), *injection attacks*, atau pencurian identitas. Dengan semakin tingginya volume data yang di*transfer* antar-layanan dan melalui jaringan, keamanan REST API menjadi aspek krusial untuk menjamin integritas dan kerahasiaan data, serta

untuk memastikan tidak ada layanan yang disusupi oleh pihak tidak bertanggung jawab. Karena konsep dari REST API adalah bahwa *server* menyediakan sumber daya yang dapat diakses dan dimanipulasi oleh klien melalui permintaan HTTP (*HyperText Transfer Protocol*). Setiap sumber daya direpresentasikan dalam bentuk URI (*Uniform Resource Identifier*) yang unik. Melalui metode HTTP seperti *GET, POST, PUT,* dan *DELETE*, klien dapat berinteraksi dengan sumber daya tersebut untuk melakukan operasi seperti membaca, menambah, memperbarui, atau menghapus data. (Febriansyah, 2023).

Arsitektur microservice telah mendapatkan popularitas dalam beberapa tahun terakhir karena kemampuannya untuk memecah aplikasi kompleks menjadi komponen yang lebih kecil dan lebih mudah dikelola. (Harshavardhan & Julia, 2021) Alat orkestrasi *container* seperti *Kubernetes* dan Docker Swarm sangat penting untuk mengelola dan menerapkan *microservice* dalam skala besar. Kontainer ringan digunakan untuk pemanfaatan sumber daya *cloud* yang efisien dengan penerapan secara bersamaan. Kebijakan penskalaan otomatis menjadi fokus untuk meminimalkan biaya pemrosesan dan pemanfaatan sumber daya yang efisien di pusat data. Strategi penskalaan otomatis berbasis kontainer yang diusulkan bertujuan untuk meminimalkan biaya operasional dan memenuhi berbagai persyaratan QoS dengan menskalakan sumber daya fisik berdasarkan kebutuhan dan ketersediaan sumber daya *microservice*. (Mohd et al., 2022)(Satish et al., 2020) Mereka juga menawarkan fitur-fitur seperti penyeimbangan beban, penskalaan otomatis, dan kemampuan penyembuhan mandiri, yang membantu mengoptimalkan kinerja dan keandalan *microservice*. Teknik penskalaan otomatis dalam komputasi *cloud* dan edge berbasis kontainer menawarkan fitur-fitur seperti penyeimbangan beban, penyeimbangan beban otomatis, dan komputasi *cloud*. penskalaan, dan kemampuan penyembuhan mandiri untuk mengoptimalkan kinerja dan keandalan *microservice*. Teknik-teknik ini secara dinamis menyesuaikan sumber daya yang dialokasikan berdasarkan permintaan, memastikan pemanfaatan sumber daya yang efisien dan kualitas layanan selama jam sibuk. (Reza & Farshad, 2023)(ΓΙ, 2024). Dengan memanfaatkan alat orkestrasi kontainer, pengembang dapat dengan mudah menerapkan, memperbarui, dan memantau aplikasi mereka, sehingga menghasilkan

siklus pengembangan yang lebih cepat dan meningkatkan efisiensi secara keseluruhan. Dengan memanfaatkan alat orkestrasi *container* seperti *Kubernetes*, pengembang dapat menentukan keadaan lingkungan *container* yang diinginkan dan mengotomatiskan proses penerapan, penskalaan, dan langkah-langkah keamanan. Hal ini memungkinkan siklus pengembangan lebih cepat dan memastikan aplikasi selalu tersedia dan responsif. (Vamsikrishna. et al., 2021) Seiring dengan meningkatnya permintaan *microservice*, peran alat orkestrasi *container* akan menjadi semakin penting dalam praktik pengembangan perangkat lunak modern. Alat orkestrasi *container* seperti *Kubernetes,* Docker Swarm, dan layanan *container* AWS menjadi semakin penting dalam praktik pengembangan perangkat lunak modern karena meningkatnya permintaan akan *microservice*. Alat-alat ini memecahkan masalah-masalah utama dengan memungkinkan alur penerapan yang cepat dan strategi penerapan perangkat lunak tingkat lanjut, memastikan keberhasilan *eksekusi* di lingkungan yang berbeda, dan memungkinkan penerapan beberapa lingkungan *eksekusi* menggunakan rantai alat yang sama. Rekayasa dan penerapan perangkat lunak yang berkelanjutan, yang difasilitasi oleh alat otomatisasi, mendapatkan perhatian dan dipandang sebagai gelombang besar berikutnya dalam rekayasa perangkat lunak setelah gerakan pengembangan perangkat lunak yang tangkas dan ramping. (Mohammad dkk., 2019)

Load balancing dan *microservice* adalah dua konsep yang berkembang pesat di dunia teknologi informasi, khususnya dalam bidang pengembangan aplikasi dan web. *Load balancing* dan *mikroservice* saling berkaitan karena keduanya bertujuan untuk meningkatkan kinerja, *Skalabilitas*, dan reliabilitas aplikasi dan web yang berbasis *arsitektur* ter*distribusi*. *Load balancing* adalah proses men*distribusi*kan beban permintaan ke beberapa *server* atau instansi yang melayani aplikasi atau web. *Load balancing* dapat dilakukan dengan menggunakan perangkat keras (hardware) atau perangkat lunak (software) yang disebut *load balancer*. *Load balancer* bertugas untuk memilih *server* atau instance yang paling sesuai untuk menerima permintaan dari klien, berdasarkan kriteria tertentu seperti ketersediaan, kapasitas, lokasi, konten, dan lain-lain. *Load balancing* dapat meningkatkan performa, ketersediaan,

dan efisiensi aplikasi atau web dengan cara menghindari kelebihan beban atau kegagalan pada *server* atau instance tertentu.

Microservice adalah *arsitektur* aplikasi atau web yang terdiri dari sekumpulan *microservice* (*microservice*) yang saling berkomunikasi melalui *protokol* ringan. *Microservice* memiliki karakteristik seperti *modularitas, otonomi, desentralisasi*, dan *heterogenitas*. *Microservice* dapat meningkatkan *Skalabilitas, fleksibilitas,* dan *reliabilitas* aplikasi atau web dengan cara memungkinkan penambahan atau pengurangan layanan sesuai dengan kebutuhan, memisahkan tanggung jawab dan fungsi antara layanan, mengurangi ketergantungan dan keterkaitan antara layanan, dan memanfaatkan teknologi yang berbeda-beda untuk setiap layanan.Seiring dengan berkembangnya *arsitektur* perangkat lunak, kemunculan *arsitektur microservice* menjadi sebuah tren yang digunakan oleh pengembang dalam beberapa tahun terakhir. *Microservice* memberikan keleluasaan bagi pengembang untuk mengembangkan perangkat lunak dalam waktu yang cepat. Berkembangnya *arsitektur microservice* juga didukung oleh kurang handalnya *arsitektur* monolith dalam menangani *system failure*. Karena dalam satu aplikasi 2 memiliki satu codebase yang sama, merupakan hal yang pasti bahwa *arsitektur* monolith juga menjadi single point of failure yang akan terjadi jika salah satu layanan di codebase yang sama mengalami malfungsi atau *error*. Sementara itu, *arsitektur microservice* memungkinkan sebuah sistem dibangun oleh codebase yang berbeda-beda atau layanan yang terpisah sehingga dapat menawarkan sistem yang resilien dan tetap menjamin *availability* ketika salah satu layanan mengalami malfungsi. Keunggulan *arsitektur microservice* lainnya adalah perubahan yang terjadi pada satu layanan dalam frekuensi yang tinggi akan tetap membuat sistem secara keseluruhan bekerja dengan baik tanpa adanya perubahan yang signifikan. Keunggulan *arsitektur microservice* juga ada pada *point scale*. Dengan menggunakan *arsitektur microservice*, scale dapat dilakukan pada sistem atau komponen yang membutuhkan scale saja, tanpa harus scale pada keseluruhan sistem.

Sebuah program aplikasi (aplikasi perangkat lunak, atau aplikasi, atau aplikasi singkatnya program komputer yang dirancang untuk melaksanakan tugas tertentu

selain yang berkaitan dengan pengoperasian komputer itu sendiri, biasanya untuk digunakan oleh pengguna akhir. Pengolah kata, pemutar media, dan perangkat lunak akuntansi adalah contohnya. Kata benda kolektif "perangkat lunak aplikasi" mengacu pada semua aplikasi secara kolektif. Klasifikasi utama perangkat lunak lainnya adalah perangkat lunak sistem, yang berkaitan dengan pengoperasian komputer, dan perangkat lunak utilitas ("utilitas").

Aplikasi dapat dipaketkan dengan komputer dan perangkat lunak sistemnya atau diterbitkan secara terpisah dan dapat dikodekan sebagai hak milik, sumber terbuka, atau proyek. Istilah "aplikasi" biasanya mengacu pada aplikasi untuk perangkat seluler seperti ponsel Penskalaan, juga dieja penskalaan otomatis atau otomatis, dan terkadang juga disebut penskalaan otomatis, adalah metode yang digunakan dalam komputasi awan yang secara dinamis menyesuaikan jumlah sumber daya komputasi dalam kumpulan *server* - biasanya diukur dengan jumlah *server* aktif - secara otomatis berdasarkan beban di perkebunan. Misalnya, jumlah *server* yang berjalan di belakang aplikasi web dapat ditambah atau dikurangi secara otomatis berdasarkan jumlah pengguna aktif di situs. Karena metrik tersebut dapat berubah secara dramatis sepanjang hari, dan *server* adalah sumber daya terbatas yang memerlukan biaya untuk dijalankan bahkan saat menganggur, sering kali ada insentif untuk menjalankan *server* yang "cukup" untuk mendukung beban saat ini sambil tetap dapat melakukannya. mendukung lonjakan aktivitas yang tiba-tiba dan besar. Penskalaan otomatis berguna untuk kebutuhan tersebut, karena dapat mengurangi jumlah *server* aktif saat aktivitas rendah, dan meluncurkan *server* baru saat aktivitas tinggi. Penskalaan otomatis terkait erat dengan, dan dibangun di atas, gagasan penyeimbangan beban.

Arsitektur mikro – varian dari SOA gaya struktural pola *arsitektur* yang mengatur aplikasi sebagai kumpulan digabungkan secara longgar yang berbutir halus, berkomunikasi melalui *protokol* ringan. Salah satu tujuannya adalah agar tim dapat mengembangkan dan menggunakan layanan mereka secara *independen* dari yang lain. Hal ini dicapai dengan pengurangan beberapa dependensi dalam basis kode, yang memungkinkan pengembang mengembangkan layanan mereka dengan batasan

terbatas dari pengguna, dan bagi pengguna untuk disembunyikan dari kerumitan tambahan. Akibatnya, organisasi dapat mengembangkan perangkat lunak dengan pertumbuhan dan ukuran yang cepat, serta menggunakan layanan siap pakai dengan lebih mudah. Persyaratan komunikasi berkurang. Manfaat ini datang dengan biaya untuk mempertahankan decoupling. Antarmuka perlu dirancang dengan hati-hati dan diperlakukan sebagai API publik. Salah satu teknik yang digunakan adalah memiliki banyak antarmuka pada layanan yang sama, atau beberapa versi dari layanan yang sama, agar tidak mengganggu pengguna kode yang ada.

Alat-alat ini memberikan tingkat otomatisasi dan kontrol yang penting untuk mengelola kompleksitas penerapan dan penskalaan *microservice* dalam lingkungan yang dinamis dan terus berubah. Alat yang disebutkan dalam sumber memberikan penskalaan hierarki dan opsi kontrol terdesentralisasi untuk mengelola *microservice* di *Kubernetes*. Alat ini bertujuan untuk meningkatkan efisiensi dan *Skalabilitas* aplikasi dengan mengotomatiskan penyediaan, pengelolaan, komunikasi, dan toleransi kesalahan kontainer. Alat tersebut juga memperkenalkan dua- *arsitektur* kontrol hirarki berlapis untuk mengadaptasi aplikasi berbasis *microservice*, dengan komponen terdesentralisasi yang meng*implementasi*kan *loop MAPE local*. Alat ini berfokus pada adaptasi proaktif berdasarkan perkiraan beban dan estimasi waktu respons untuk mengoptimalkan penerapan aplikasi. (Valeria & Francesco, n.d.) Selain itu, alat orkestrasi *container* memungkinkan pengembang untuk mengabstraksi *infrastruktur* yang mendasarinya, memungkinkan mereka untuk fokus dalam membangun dan meningkatkan aplikasi mereka tanpa harus khawatir tentang seluk-beluk pengelolaan *container* dan layanan. Tingkat abstraksi ini juga memudahkan tim untuk berkolaborasi dan bekerja sama secara lancar, terlepas dari keahlian atau latar belakang masing-masing. Pada akhirnya, alat *orkestrasi container* memainkan peran penting dalam memungkinkan organisasi memanfaatkan potensi penuh *microservice* dan mencapai ketangkasan, *Skalabilitas*, dan keandalan yang lebih baik dalam proses pengembangan perangkat lunak mereka.

Dengan menyederhanakan penerapan dan penskalaan kontainer, alat-alat ini membantu organisasi merespons dengan cepat terhadap perubahan permintaan pasar

dan kebutuhan pelanggan. Pipa CI/CD di *Kubernetes* menyederhanakan penerapan dan penskalaan *container*, memungkinkan organisasi merespons dengan cepat terhadap perubahan permintaan pasar dan kebutuhan pelanggan. Dengan mengotomatiskan proses pengujian, pemantauan, dan penerapan, alat ini memungkinkan pengembang untuk mengidentifikasi dan mengatasi masalah sejak dini, mempromosikan perbaikan berkelanjutan dan peningkatan kualitas perangkat lunak. *Skalabilitas Kubernetes* memastikan bahwa *pipeline* CI/CD dapat menangani berbagai beban kerja secara efisien, mengoptimalkan pemanfaatan sumber daya dan efisiensi biaya. Integrasi *pipeline CI/CD* dengan *Kubernetes* juga meningkatkan keamanan dengan mendeteksi dan mengatasi potensi kerentanan sejak dini. proses pengembangan, yang pada akhirnya meningkatkan postur keamanan aplikasi secara keseluruhan. (UDH., 2022) Ketangkasan ini sangat penting dalam lingkungan bisnis yang serba cepat saat ini, di mana kemampuan beradaptasi dan berinovasi dengan cepat dapat menjadi penentu antara kesuksesan dan kegagalan. Dengan alat orkestrasi *container*, tim dapat menerapkan fitur-fitur baru dan pembaruan dengan percaya diri, mengetahui bahwa *infrastruktur* yang mendasarinya dikelola secara efisien dan efektif. (Dimitri et al., 2019)(Vamsikrishna. et al., 2021). Hal ini tidak hanya memungkinkan waktu pemasaran produk dan layanan baru yang lebih cepat, namun juga memastikan pengalaman pengguna yang lancar dan andal. Selain itu, *Skalabilitas* yang disediakan oleh alat orkestrasi kontainer memungkinkan organisasi menangani peningkatan lalu lintas dan beban kerja dengan mudah tanpa mengorbankan kinerja. (Vamsikrishna. et al., 2021) Keandalan alat ini memastikan aplikasi berjalan dengan lancar dan konsisten, meminimalkan waktu henti dan memaksimalkan efisiensi. Secara keseluruhan, penggunaan alat orkestrasi *container* memungkinkan organisasi untuk tetap kompetitif dalam lanskap pasar yang terus berkembang.

Komputasi awan memberikan peluang baru untuk menerapkan aplikasi yang dapat diskalakan dengan cara yang efisien, memungkinkan aplikasi perusahaan menyesuaikan sumber daya komputasinya secara dinamis sesuai permintaan. Dalam makalah ini kami menganalisis dan menguji pola *arsitektur microservice*, yang digunakan selama beberapa tahun terakhir oleh perusahaan *Internet* besar seperti

Amazon, Netflix, dan *LinkedIn* untuk menerapkan aplikasi besar di *cloud* sebagai serangkaian *microservice* yang dapat dikembangkan, diuji, diterapkan, ditingkatkan skalanya., dioperasikan dan ditingkatkan secara *independen*, memungkinkan perusahaan-perusahaan ini memperoleh kehandalan, mengurangi kompleksitas, dan menskalakan aplikasi mereka di *cloud* dengan cara yang lebih efisien. Kami menyajikan studi kasus di mana aplikasi perusahaan dikembangkan dan diterapkan di *cloud* menggunakan pendekatan *monolitik* dan *arsitektur microservice* menggunakan kerangka *web deploy*. Kami menunjukkan hasil uji kinerja yang dijalankan pada kedua aplikasi, dan kami menjelaskan manfaat dan tantangan yang dapat diperoleh dan dihadapi oleh perusahaan saat ini ketika mereka menerapkan *microservice* dalam aplikasi mereka (M. Villamizar, O. Garcer, H. Castro et all, 2015)

Ide di balik *arsitektur microservice* adalah untuk mengembangkan aplikasi tunggal yang besar dan kompleks sebagai rangkaian *microservice*, *kohesif*, dan *independen*. Di sisi lain, sistem *monolitik* menjadi semakin besar seiring berjalannya waktu, menyimpang dari *arsitektur* yang diharapkan, serta menjadi berisiko dan mahal untuk dikembangkan. Makalah ini menjelaskan teknik untuk mengidentifikasi dan mendefinisikan *microservice* pada sistem perusahaan *monolitik*. Sebagai kontribusi besar, evaluasi kami menunjukkan bahwa pendekatan kami mampu mengidentifikasi kandidat yang relevan untuk menjadi *microservice* pada sistem (A. Levcovit, R. Terra, M. Valente, 2016).

Komputasi awan telah memungkinkan penyediaan layanan yang elastis dan sesuai permintaan untuk mencapai pemanfaatan sumber daya yang lebih efisien dan respons yang lebih cepat terhadap berbagai beban aplikasi. Mesin *virtual*, yang merupakan elemen penyusun *cloud*, dapat dibuat menggunakan templat khusus penyedia yang disimpan dalam *repositori* berpemilik, yang dapat mengakibatkan terkuncinya penyedia dan penurunan portabilitas. Meskipun terdapat teknologi yang memungkinkan, aplikasi berorientasi layanan berskala besar masih bersifat *inrealastis*. Aplikasi semacam itu sering kali menggunakan layanan *monolitik* yang membatasi elastisitasnya (misalnya, dengan menghalangi replikasi bagian dari

layanan *monolitik*). Penguraian layanan-layanan ini (mengarah ke layanan yang lebih kecil, lebih bertarget, dan lebih *modular*) akan mengarah pada elastisitas, namun proses penguraiannya sebagian besar dilakukan secara manual. Makalah ini memperkenalkan metodologi untuk menguraikan layanan *monolitik* menjadi beberapa *microservice*. Metodologi yang diusulkan menerapkan beberapa hasil proyek *ENTICE* (yaitu alat sintesis dan pengoptimalan gambar). Terakhir, makalah ini memberikan wawasan tentang bagaimana hasil ini membantu merevitalisasi layanan *monolitik* di masa lalu, dan teknik apa yang diterapkan untuk membantu pengembang *microservice* di masa depan (G, Kecskemeti, A. Marosi, A. Kertesz, 2016).

Arsitektur microservice mendapat perhatian dari komunitas akademis dan *industri*, dan sebagian besar dibandingkan dengan *arsitektur monolitik*. Banyak hasil makalah penelitian ini yang saling bertentangan mengenai kinerja *arsitektur* tersebut. Oleh karena itu, kedua *arsitektur* ini dibandingkan dalam makalah ini, dan beberapa konfigurasi spesifik aplikasi *microservice* juga dievaluasi dalam istilah penemuan layanan. *Arsitektur monolitik* dalam pengujian konkurensi menunjukkan kinerja *throughput* yang lebih baik sebesar 6% jika dibandingkan dengan *arsitektur microservice*. Skenario pengujian beban tidak menunjukkan perbedaan yang signifikan antara kedua *arsitektur*. Selain itu, pengujian ketiga yang membandingkan aplikasi *microservice* yang dibangun dengan teknologi penemuan layanan berbeda seperti Konsul dan Eureka menunjukkan bahwa aplikasi dengan Konsul memberikan hasil yang lebih baik dalam hal *throughput* (O. al-Debagy, P. Martinek, 2018).

Muncul dari komunitas praktisi yang tangkas, *arsitektur* berorientasi *microservice* menekankan penerapan dan penggunaan beberapa *microservice* berskala kecil dan dapat diterapkan secara *independen*, dibandingkan merangkum semua kemampuan fungsi ke dalam satu aplikasi *monolitik*. Sejalan dengan itu, dekomposisi berorientasi *microservice*, yang telah diidentifikasi sebagai tugas yang sangat menantang dan kompleks, memainkan peran penting dan prasyarat dalam pengembangan sistem perangkat lunak berbasis *microservice*. Untuk mengatasi

tantangan ini dan mengurangi kompleksitas, kami mengusulkan pendekatan *analisis top-down* dan mengembangkan *algoritma* dekomposisi berbasis aliran data. Secara singkat, proses tiga langkah didefinisikan: pertama, insinyur bersama dengan pengguna melakukan analisis kebutuhan bisnis dan membangun diagram aliran data logika bisnis yang murni dan terperinci; kemudian, *algoritma* kami menggabungkan operasi yang sama dengan tipe data keluaran yang sama ke dalam aliran data abstrak *virtual*; terakhir, *algoritma* mengekstrak masing-masing modul 'operasi dan data keluarannya' dari aliran data *abstrak virtua*l untuk mewakili kandidat *microservice* yang teridentifikasi. Kami telah menggunakan dua kasus penggunaan untuk mendemonstrasikan mekanisme identifikasi *microservice* kami, serta membuat perbandingan dengan alat identifikasi *microservice* yang ada. Perbandingan dan *evaluasi* menunjukkan bahwa, mekanisme identifikasi berbasis aliran data kami mampu menghasilkan kandidat *microservice* yang lebih rasional, objektif, mudah dipahami, dan konsisten, melalui prosedur *implementasi* yang lebih ketat dan praktis (R. Chen, S.Li, Z. Li, 2018).

Muncul dari komunitas praktisi yang tangkas, *arsitektur* ber*orientasi microservice* menekankan penerapan dan penggunaan beberapa *microservice* berskala kecil dan dapat diterapkan secara *independen*, dibandingkan merangkum semua kemampuan fungsi ke dalam satu aplikasi *monolitik*. Sejalan dengan itu, dekomposisi berorientasi *microservice*, yang telah diidentifikasi sebagai tugas yang sangat menantang dan kompleks, memainkan peran penting dan prasyarat dalam pengembangan sistem perangkat lunak berbasis *microservice*. Untuk mengatasi tantangan ini dan mengurangi kompleksitas, kami mengusulkan pendekatan analisis *top-down* dan mengembangkan algoritma dekomposisi berbasis aliran data. Secara singkat, proses tiga langkah didefinisikan: pertama, *insinyur* bersama dengan pengguna melakukan analisis kebutuhan bisnis dan membangun diagram aliran data logika bisnis yang murni dan terperinci; kemudian, algoritma kami menggabungkan operasi yang sama dengan tipe data keluaran yang sama ke dalam aliran data *abstrak virtual*; terakhir, algoritma mengekstrak masing-masing modul '*operasi* dan data keluarannya' dari aliran data *abstrak virtual* untuk mewakili kandidat *microservice* yang teridentifikasi. Kami telah menggunakan dua kasus penggunaan untuk

mendemonstrasikan mekanisme identifikasi *microservice* kami, serta membuat perbandingan dengan alat identifikasi *microservice* yang ada. Perbandingan dan evaluasi menunjukkan bahwa, mekanisme identifikasi berbasis aliran data kami mampu menghasilkan kandidat *microservice* yang lebih *rasional*, *objektif,* mudah dipahami, dan konsisten, melalui prosedur *implementasi* yang lebih ketat dan praktis (F. Ponce, G. Marquez, H. Astudillo, 2019).

Dengan memanfaatkan alat *orkestrasi container,* organisasi dapat menyederhanakan proses pengembangan dan penerapannya, sehingga memungkinkan mereka merespons permintaan pasar dengan cepat dan efisien. Alat-alat ini memberikan tingkat otomatisasi yang menyederhanakan pengelolaan aplikasi ter*kontainer* yang kompleks, sehingga memudahkan tim untuk berkolaborasi dan berinovasi. Alat *orkestrasi container* otomatis memberikan penyeimbangan beban, penskalaan otomatis, dan otomatisasi penerapan, menyederhanakan pengelolaan aplikasi dalam *container* yang kompleks dan memungkinkan tim untuk berkolaborasi dan berinovasi. (Vamsikrishna. et al., 2021) Dengan kemampuan untuk meningkatkan atau menurunkan skala sumber daya dengan mudah sesuai kebutuhan, organisasi dapat beradaptasi terhadap perubahan kebutuhan bisnis tanpa gangguan apa pun pada operasi mereka. Fleksibilitas dan ketangkasan ini sangat penting dalam lingkungan bisnis yang bergerak cepat saat ini, di mana kemampuan untuk melakukan pivot dan melakukan iterasi dengan cepat dapat menjadi penentu antara kesuksesan dan kegagalan. Dengan memanfaatkan alat-alat ini, organisasi dapat tetap menjadi yang terdepan dalam persaingan dan mempertahankan keunggulan kompetitif di pasar. Kemampuan untuk menerapkan fitur dan pembaruan baru dengan cepat memungkinkan perusahaan memenuhi permintaan pelanggan dan tetap relevan dalam lanskap yang terus berkembang. Secara keseluruhan, fleksibilitas dan ketangkasan yang diberikan oleh alat otomatisasi ini sangat penting bagi bisnis untuk berkembang dalam dunia bisnis yang dinamis dan kompetitif saat ini. Alat otomatisasi menyederhanakan alur kerja dan meningkatkan efisiensi. Mereka juga memungkinkan bisnis untuk beradaptasi dengan cepat terhadap perubahan di pasar dan. meningkatkan kinerja mereka secara keseluruhan. Alat otomasi dapat menyederhanakan alur kerja dan meningkatkan

efisiensi dengan menghemat waktu dan meningkatkan akurasi, seperti yang terlihat dalam penggunaan alat otomasi tinjauan sistematis dalam pelayanan kesehatan. Kurangnya pengetahuan tentang alat ini merupakan hambatan umum dalam penerapannya. (Tracy & Steven, 2023)(Anna dkk., 2021)

Menguji kinerja dan efisiensi dalam aplikasi *microservice* sangat penting untuk memastikan bahwa sistem ini dapat menangani tuntutan operasi bisnis modern. Dengan melakukan pengujian kinerja secara menyeluruh, organisasi dapat mengidentifikasi hambatan atau masalah apa pun yang mungkin memengaruhi keseluruhan fungsi *arsitektur microservice* mereka. Pendekatan pengujian yang proaktif ini memungkinkan perusahaan untuk mengatasi potensi masalah sebelum masalah tersebut memengaruhi pengalaman pengguna atau menyebabkan downtime. Selain itu, mengukur efisiensi aplikasi *microservice* dapat membantu organisasi mengoptimalkan sumber daya mereka dan memastikan bahwa mereka beroperasi pada tingkat kinerja puncak. Dengan terus memantau dan menguji kinerja dan efisiensi *microservice* mereka, perusahaan dapat mempertahankan tingkat keandalan yang tinggi dan memberikan pengalaman pengguna yang lancar kepada pelanggan mereka. Tingkat keandalan dan kinerja ini sangat penting dalam pasar yang kompetitif saat ini, di mana pengalaman pengguna dapat menentukan atau menghancurkan reputasi perusahaan. Dengan tetap mengikuti pengujian dan pemantauan, organisasi dapat mengatasi segala masalah yang mungkin timbul dan terus memberikan layanan terbaik kepada pelanggan mereka. Pada akhirnya, dengan memprioritaskan efisiensi dan keandalan *arsitektur microservice* mereka, perusahaan dapat mempersiapkan diri untuk mencapai kesuksesan dan pertumbuhan jangka panjang dalam lanskap digital yang terus berkembang. Dengan pesatnya kemajuan teknologi dan meningkatnya permintaan konsumen, perusahaan harus terus beradaptasi dan meningkatkan *arsitektur microservice* mereka untuk memenuhi kebutuhan penggunanya. Dengan berinvestasi dalam pengujian dan pemantauan berkelanjutan, organisasi dapat memastikan bahwa sistem mereka berjalan lancar dan efisien, yang pada akhirnya menghasilkan kepuasan dan loyalitas pelanggan yang lebih tinggi. Di dunia yang serba cepat saat ini, perusahaan yang

memprioritaskan keandalan dan kinerja *arsitektur microservice* mereka akan lebih siap untuk berkembang dan sukses di pasar yang kompetitif.(Peng et al., 2021) 1. Jelajahi manfaat berinvestasi dalam pengujian dan pemantauan berkelanjutan untuk *arsitektur microservice*, termasuk bagaimana hal ini dapat meningkatkan kinerja sistem secara keseluruhan dan kepuasan pelanggan. 2. Membahas strategi spesifik yang dapat diterapkan perusahaan untuk mengadaptasi dan meningkatkan *arsitektur microservice* mereka sebagai respons terhadap kemajuan teknologi yang pesat dan perubahan permintaan konsumen. 3. Memeriksa studi kasus perusahaan yang telah berhasil memprioritaskan keandalan dan kinerja dalam *arsitektur microservice* mereka, dengan menyoroti dampaknya terhadap keunggulan kompetitif mereka di pasar. 4. Menganalisis potensi tantangan yang mungkin dihadapi organisasi ketika mencoba meningkatkan atau mengoptimalkan *arsitektur microservice* mereka, seperti masalah integrasi atau masalah *Skalabilitas*. 5. Pertimbangkan tren masa depan dalam teknologi dan perilaku konsumen yang selanjutnya dapat membentuk evolusi *arsitektur microservice* dan diskusikan bagaimana perusahaan dapat mempersiapkan diri menghadapi perubahan ini agar tetap menjadi yang terdepan dalam persaingan. (Wolff, n.d.) (Di et al., 2019)

Salah satu contoh perusahaan yang berhasil memprioritaskan keandalan dan kinerja dalam *arsitektur microservice*nya adalah Netflix. Dengan memanfaatkan *infrastruktur microservice* yang sangat toleran terhadap kesalahan dan terukur, Netflix mampu memberikan pengalaman streaming yang lancar kepada jutaan pengguna di seluruh dunia, sehingga memberi mereka keunggulan kompetitif yang signifikan di pasar streaming yang ramai. Namun, tantangan mungkin muncul ketika organisasi mencoba meningkatkan atau mengoptimalkan *arsitektur microservice* mereka, seperti memastikan integrasi yang lancar antara berbagai layanan atau melakukan penskalaan secara efisien untuk memenuhi permintaan yang semakin meningkat. Salah satu contoh tandingan dari keberhasilan Netflix dalam memprioritaskan keandalan dan kinerja adalah kasus pemadaman listrik besar-besaran yang mereka alami pada tahun 2019, yang berdampak pada ribuan pengguna dan mengakibatkan hilangnya pendapatan secara signifikan. Selain itu, meskipun Netflix memiliki *infrastruktur microservice* yang kuat, organisasi lain mungkin

kesulitan dalam menerapkan dan memelihara sistem yang rumit tersebut, sehingga berpotensi menimbulkan masalah kinerja dan waktu henti.

Perkembangan *load balancing* dan *mikroservice* di dunia dipengaruhi oleh beberapa faktor seperti meningkatnya permintaan akan aplikasi atau web yang responsif, andal, dan mudah disesuaikan, kemajuan teknologi *cloud computing* yang menyediakan *platform* untuk menjalankan *load balancing* dan mikroservice secara efisien dan ekonomis, serta adopsi dari perusahaan-perusahaan besar seperti Netflix, Amazon, Google, Facebook, dll yang menggunakan *load balancing* dan *microservice* untuk mendukung operasional mereka. Beberapa contoh aplikasi atau web yang menggunakan *load balancing* dan *mikroservice* di dunia antara lain adalah Spotify, Uber, Airbnb, Twitter, Instagram, dll.

Kelebihan dan kekurangan *load balancing* dan *autoscaling microservices* architecture adalah sebagai berikut, Kelebihan: *Load balancing* dapat meningkatkan performa, ketersediaan, dan efisiensi aplikasi atau web dengan cara menghindari kelebihan beban atau kegagalan pada *server* atau instansi tertentu. *Autoscaling* dapat membantu aplikasi atau web untuk menangani lonjakan permintaan saat beban tinggi, dan menghemat sumber daya saat beban rendah. *Microservices* dapat meningkatkan *Skalabilitas*, *fleksibilitas*, dan *reliabilitas* aplikasi atau web dengan cara memungkinkan penambahan atau pengurangan layanan sesuai dengan kebutuhan, memisahkan tanggung jawab dan fungsi antara layanan, mengurangi ketergantungan dan keterkaitan antara layanan, dan memanfaatkan *teknologi* yang berbeda-beda untuk setiap layanan. *Arsitektur terdistribusi* dapat meningkatkan performa, *Skalabilitas*, dan reliabilitas aplikasi atau web dengan cara memanfaatkan sumber daya yang lebih banyak dan lebih dekat dengan klien, serta dengan cara menangani kegagalan pada komponen tertentu tanpa mengganggu komponen lainnya.

Kekurangan pada *Load balancing* membutuhkan perangkat keras (*hardware*) atau perangkat lunak (*software*) yang disebut load balancer, yang dapat menambah biaya dan kompleksitas dalam pengelolaan aplikasi atau web. *Autoscaling* membutuhkan dukungan dari *platform cloud*, seperti AWS, Azure, atau *Google*

Cloud Platform, yang dapat menambah biaya dan ketergantungan pada penyedia layanan *cloud*. *Microservices* membuat aplikasi atau web menjadi lebih kompleks dan padat, sehingga membutuhkan koordinasi dan komunikasi yang lebih baik antara tim pengembang, serta membutuhkan alat-alat tambahan untuk melakukan *service discovery, centralized configuration, API gateway, centralized logging,* dan lain lain. *Arsitektur* ter*distribusi* membuat aplikasi atau web menjadi lebih rentan terhadap masalah penerapan efisinsi, performa *microservice*, skala *cluster* dan lain lain, sehingga membutuhkan *protokol* dan mekanisme yang lebih baik untuk mengatasi masalah-masalah tersebut.

B. Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana *implementasi* pengujian performa dan efisiensi pada *microservice* pada *Kubernetes* k0s, k3s, k3d dan k8s.
2. Tahapan apa saja yang diperlukan dan dilakukan dalam *implementasi microservice* pada *Kubernetes* k0s, k3s, k3d dan k8s.
3. Bagaimana perbandingan ukuran efisiensi dan kecepatan performa aplikasi *microservice* pada *Kubernetes* k0s, k3s, k3d dan k8s.

C. Batasan Masalah

Batasan masalah merupakan ruang lingkup dari permasalahan yang akan dibatasi dalam penelitian mengenai *implementasi* pengujian performa dan efisiensi pada *microservice microservice* pada *Kubernetes* k0s, k3s, k3d dan *Kubernetes* k8s pada Mahasiswa Informatika maupun masyarakat umum dan system administrator lain. Adapun Batasan masalah yang sudah ditetapkan dalam penelitian ini, antara lain:

1. Penelitian ini tertuju kepada mahasiswa maupun masyarakat umum yang sedang mengembangkan aplikasi virtualisasi *microservice*.
2. Penelitian ini menguji ukuran efisiensi dan kecepatan performa aplikasi *microservice webserver* pada *Kubernetes* k0s, k3s, k3d dan *Kubernetes* k8s.

3. *Parameter* yang diuji adalah tingkat efisiensi dan performa kecepatan dari praktikum yang diberikan dengan beberapa faktor seperti pengembangan *independen*, penerapan *devops*, penskalaan *cluster*, alat *manajemen*, pengujian performa.
4. Penelitian ini termasuk dalam penelitian kualititatif dan teknik pengumpulan datanya menggunakan pendekatan *experimental*.
5. Penelitian ini juga membuat produk *microservice* menggunakan perbandingan antara *microservice webserver* pada *Kubernetes* k0s, k3s, k3d dan *Kubernetes* k8s.

D. Tujuan Penelitian

Berdasarkan rumusan masalah dan Batasan masalah di atas, maka tujuan dari penelitian ini adalah melakukan pengukuran efisiensi dan performa kecepatan pada proses pengembangan aplikasi *microservice* yang sangat mungkin berisi data sensitif dengan metode kuantitatif dan teknik pengumpulan datanya menggunakan pendekatan *experimental.*

E. Manfaat Penelitian

Dalam hasil penelitian yang telah dilakukan, diharapkan akan memberikan manfaat sebagai berikut:

1. Memberikan informasi terhadap hasil *implementasi* pengujian performa dan efisiensi pada *microservice webserver* pada *Kubernetes* k0s, k3s, k3d dan *Kubernetes* k8s.
2. Meningkatkan efisiensi dan performa kecepatan pada *microservice webserver* pada *Kubernetes* k0s, k3s, k3d dan *Kubernetes* k8s
3. Memberikan data perbandingan peningkatan efisiensi dan performa kecepatan yang dibutuhkan dalam industri dan teknologi *microservice* web*server*.

BAB V

PENUTUP

A. Kesimpulan

Berdasarkan hasil penelitian yang telah dilakukan, dapat ditarik kesimpulan bahwa Penerapan k3s dalam konteks *microservices* web *server* terbukti memberikan efisiensi dan performa yang lebih baik. Efisiensi Sumber Daya *Distribusi* seperti k3s hanya membutuhkan 1 core CPU, 512 MB RAM, dan 5 GB penyimpanan untuk menjalankan *cluster all-in-one* (*Control Plane+ Worker node*), sementara k8s memerlukan setidaknya 2 *core* CPU, 2 GB RAM, dan 20 GB penyimpanan untuk *Control Plane*saja. Selain itu, k3d dan k0s juga menunjukkan efisiensi serupa dengan kebutuhan resource yang lebih rendah dibandingkan k8s. Hal ini membuat k3s, k3d, dan k0s sangat cocok untuk *edge computing*, IoT, atau pengembangan lokal di mana sumber daya komputasi terbatas.

Instalasi dan penggunaan k3s dirancang sebagai *distribusi lightweight Kubernetes* dengan *minimal dependency*, sehingga instalasi dan konfigurasinya jauh lebih sederhana dibandingkan k8s. K3d memanfaatkan Docker sebagai *backend* untuk menjalankan k3s dalam *container*, sehingga sangat ideal untuk pengembangan lokal tanpa memerlukan *setup cluster* fisik. k0s, meskipun sedikit lebih kompleks dari k3s, tetap menawarkan kemudahan instalasi dengan modularitas yang fleksibel, memungkinkan pengguna untuk menyesuaikan konfigurasi sesuai kebutuhan. Di sisi lain, K8s memerlukan konfigurasi manual yang lebih rumit, termasuk setup etcd, API *server*, scheduler, dan komponen *Control Plane*lainnya. Performa dan *Skalabilitas* pada K3s adalah *distribusi Kubernetes* yang paling efisien. K3s adalah *distribusi* yang paling efisien untuk lingkungan dengan sumber daya terbatas atau skenario *edge computing*. K3s juga sangat mudah di*instal* dan *konfigurasi*, menjadi solusi yang praktis untuk pengembangan dan produksi.

k3d adalah pilihan yang paling efisien dalam hal spesifikasi minimal dan kemampuan untuk beradaptasi dengan berbagai skenario penggunaan. Namun, pemilihan *distribusi* yang tepat tetap bergantung pada kebutuhan spesifik tim *DevOps*, karakteristik lingkungan kerja, dan skala aplikasi yang akan dijalankan. Setelah menganalisis perbandingan performa

throughput (RPS), *latency* rata-rata (ms), dan *network latency internal* (ms) untuk berbagai konfigurasi *node cluster* (1 *master + 1 Worker, 1 master + 3 Worker, 3 master + 5 Worker, dan 3 master + 10 Worker*). K3s menunjukkan performa yang lebih baik dibandingkan dengan *Kubernetes* (k8s) dalam hampir semua skenario. Namun, pilihan *distribusi* terbaik bergantung pada kebutuhan spesifik tim *DevOps*, skala aplikasi, dan lingkungan kerja. *Throughput* (RPS) pada k3s adalah *distribusi* dengan *throughput* tertinggi di semua konfigurasi *node* , mencapai hingga 8000–10000 RPS pada skala besar (3 *Master + 10 Worker*). Hal ini karena *arsitektur* minimalisnya yang dioptimalkan untuk efisiensi tinggi. K0s menunjukkan performa yang mirip dengan k3s, meskipun sedikit lebih rendah dalam beberapa kasus karena modularitasnya yang *fleksibel*. K3D menunjukkan performa yang lebih baik, tetapi *throughput*-nya sedikit lebih rendah dibandingkan K3S dan K0S karena *overhead Docker*. *Latency* Rata-Rata (ms): k3s unggul dalam hal *latency* rata-rata, dengan waktu respons terendah di semua konfigurasi *node* (misalnya, 5–15 ms pada 3 *Master* + 10 *Worker*). *Optimisasi internal* dan penggunaan SQLite sebagai database membuatnya yang tinggi. K0s menunjukkan *latency* yang sedikit lebih tinggi dibandingkan K3s (10–30 ms), tetapi masih jauh lebih dibandingkan K8s. K3d memiliki *latency* yang lebih tinggi

B. Saran

Penelitian tentang perbandingan Kubernetes (k8s), k3s, k3d, dan k0s dapat memberikan wawasan yang sangat berharga untuk memahami performa, efisiensi, dan kemampuan adaptasi masing-masing *distribusi* dalam berbagai skenario penggunaan. Salah satu saran penelitian adalah melakukan studi efisiensi sumber daya dalam skala kecil, seperti pada lingkungan *edge computing* atau IoT, dengan fokus pada evaluasi penggunaan CPU, RAM, dan penyimpanan pada konfigurasi *node* minimal (1 *Master* + 1 *Worker*). Selain itu, analisis *performa* pada *workload latency-sensitive* dapat dilakukan untuk meng*evaluasi latency* rata-rata dan *network latency internal* pada aplikasi *real-time* seperti *microservices* atau API *gateway*, menggunakan alat *benchmark* seperti k6 dan wrk atau Locust. Untuk produksi skala besar, penelitian tentang *scalability* dan *throughput* dapat membantu memahami kemampuan *distribusi* dalam menangani *workload intensif* pada skala besar (misalnya, 3 *Master* + 10 *Worker*), termasuk waktu yang diperlukan untuk *scaling up* dan *stabilitas cluster*.

Selain performa teknis, kemudahan instalasi dan konfigurasi juga penting untuk

dievaluasi, dengan mencatat waktu yang diperlukan untuk setup awal serta kompleksitas dokumentasi dan tools yang disediakan. *Efisiensi backup* dan *recovery* juga menjadi topik menarik, di mana penelitian dapat difokuskan pada pengujian proses *backup* dan *restore* data menggunakan *tools* bawaan atau pihak ketiga seperti *Velero* atau *Longhorn*, serta analisis integritas data setelah *recovery*. Studi *monitoring* dan *observability* juga dapat dilakukan untuk membandingkan kemudahan setup monitoring menggunakan tools seperti Prometheus, Grafana, atau ELK stack, serta kemampuan *distribusi* dalam menyediakan insight yang akurat tentang performa *cluster*.

Keamanan dan *compliance* juga *merupakan* aspek penting yang perlu diteliti, dengan fokus pada fitur keamanan seperti RBAC, *encryption in transit, secret management,* serta uji *vulnerability* menggunakan alat seperti *kube-bench* atau *Trivy*. Selain itu, penggunaan *distribusi* dalam lingkungan *hybrid cloud* atau *multi-cloud* dapat di*evaluasi* untuk menguji *interoperabilitas* antara *node* di berbagai *platform* cloud dan *on-premise*. Pengujian *integrasi* dengan *tools* CI/CD dan *pipeline* DevOps juga dapat dilakukan untuk membandingkan waktu yang diperlukan untuk *build, deploy, test,* dan *rollback* aplikasi, serta kemudahan penggunaan pipeline dalam masing-masing *distribusi*. Terakhir, studi tentang pemulihan dari *failure scenarios* dapat membantu memahami kemampuan *cluster* untuk pulih dari situasi seperti *node failure, pod crash,* atau jaringan terputus, dengan fokus pada waktu pemulihan dan dampak *failure* terhadap performa *cluster*.

Secara keseluruhan, penelitian ini dapat *membantu* tim DevOps membuat keputusan yang lebih tepat berdasarkan data empiris, baik untuk pengembangan lokal, *edge computing*, maupun produksi skala besar. *Distribusi* terbaik akan bergantung pada konteks penggunaan, tetapi hasil penelitian ini dapat memberikan wawasan mendalam tentang performa, efisiensi, dan kemampuan adaptasi masing-masing *distribusi* dalam berbagai skenario penggunaan.

DAFTAR PUSTAKA

id*cloud*host.com. (2021, October 7). *Load balancing* : Sejarah Singkat dan Cara Kerjanya - ID*Cloud*Host. Id*cloud*host. https://id*cloud*host.com/blog/load-balancing-sejarah-singkat-dan-cara-kerjanya/

niagahoster.com. (2021, April 7). Apa Itu *Load balancing* Pada *Server*? [Panduan Terbaru]. Niagahoster. https://www.niagahoster.co.id/blog/load-balancing-adalah/

(2023, June 2). Mengetahui *Load balancing*: Proses Penyeimbangan Traffic *Server*. Dewaweb. https://www.dewaweb.com/blog/pengertian-load-balancing/

Luo, Shutian, et al. "The power of prediction: *microservice* auto scaling via workload learning." Proceedings of the 13th Symposium on *Cloud* Computing. 2022

Blinowski, Grzegorz, Anna Ojdowska, and Adam Przybyłek. "Monolithic vs. *microservice* architecture: A performance and scalability evaluation." IEEE Access 10 (2022): 20357-20374.

Al-Debagy, Omar, and Peter Martinek. "A comparative review of *microservices* and monolithic architectures." 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI). IEEE, 2018.

Villamizar, Mario, et al. "Evaluating the monolithic and the *microservice* architecture pattern to deploy web applications in the *cloud*." 2015 10th Computing Colombian Conference (10CCC). IEEE, 2015.

Ponce, Francisco, Gastón Márquez, and Hernán Astudillo. "Migrating from monolithic architecture to *microservices*: A Rapid Review." 2019 38th International Conference of the Chilean Computer Science Society (SCCC). IEEE, 2019.

Villamizar, Mario, et al. "Infrastructure cost comparison of running web applications in the *cloud* using AWS lambda and monolithic and *microservice* architectures." 2016 16th IEEE/ACM International Symposium on *Cluster, Cloud* and Grid Computing (CCGrid). IEEE, 2016.

Fritzsch, Jonas, et al. "From monolith to *microservices*: A classification of refactoring approaches." Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and *Deployment:* First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers 1. Springer International Publishing, 2019.

Chen, Rui, Shanshan Li, and Zheng Li. "From monolith to *microservices*: A dataflow-driven approach." 2017 24th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2017.

Makris, Antonios, Konstantinos Tserpes, and Theodora Varvarigou. "Transition from monolithic to *microservice*-based applications. Challenges from the developer perspective." Open Research Europe 2 (2022): 24.

Gos, Konrad, and Wojciech Zabierowski. "The comparison of *microservice* and monolithic architecture." 2020 IEEE XVIth International Conference on the Perspective Technologies

and Methods in MEMS Design (MEMSTECH). IEEE, 2020.

Andrade, Bernardo, Samuel Santos, and António Rito Silva. "From monolith to *microservices*: Static and dynamic analysis comparison." arXiv preprint arXiv:2204.11844 (2022).

Levcovitz, Alessandra, Ricardo Terra, and Marco Tulio Valente. "Towards a technique for extracting *microservices* from monolithic enterprise systems." arXiv preprint arXiv:1605.03175 (2016).

Tapia, Freddy, et al. "From monolithic systems to *microservices*: A comparative study of performance." Applied sciences 10.17 (2020): 5797.

Kecskemeti, Gabor, Attila Csaba Marosi, and Attila Kertesz. "The ENTICE approach to decompose monolithic services into *microservices*." 2016 International Conference on High Performance Computing & Simulation (HPCS). IEEE, 2016.

Gias, Alim Ul, Giuliano Casale, and Murray Woodside. "ATOM: Model-driven *autoscaling* for *microservices*." 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2019.

Abdullah, Muhammad, et al. "Burst-aware predictive *autoscaling* for *containe*rized *microservices*." IEEE Transactions on Services Computing 15.3 (2020): 1448-1460.

Nunes, Joao Paulo KS, et al. "State of the art on *microservices autoscaling*: An overview." Anais do XLVIII Seminário Integrado de Software e Hardware (2021): 30-38.

Abdullah, Muhammad, et al. "Predictive *autoscaling* of *microservices* hosted in fog microdata center." IEEE Systems Journal 15.1 (2020): 1275-1286.

Putra, Yuri Chandra Tri. *Implementasi Arsitektur Microservice* pada Aplikasi Web Pengajaran Agama Islam Home Pesantren. Diss. Universitas Atma Jaya Yogyakarta, 2020.

"*Microservices* - Wikipedia." https://en.wikipedia.org/wiki/*Microservices*. Accessed 27 Dec. 2022.

"*Autoscaling* - Wikipedia." https://en.wikipedia.org/wiki/*Autoscaling*. Accessed 27 Dec. 2022.

"Web *server* - Wikipedia." https://en.wikipedia.org/wiki/Web_*server*. Accessed 27 Dec. 2022.

"Vertical Pod *autoscaling* | Google *Kubernetes* Engine (GKE)." https://*cloud.google.com/Kubernetes-engine/docs/concepts/verticalpodautoscaler*. Accessed 29 Dec. 2022.

"Vertical Pod Autoscaler - Amazon EKS - AWS Documentation." https://docs.aws.amazon.com/eks/latest/userguide/vertical-pod-autoscaler.html. Accessed 29 Dec. 2022.

"minikube start." 29 Nov. 2022, https://minikube.sigs.k8s.io/docs/start/. Accessed 29 Dec. 2022.

Aakash, Peng, Muhammad, Arif, Iftikhar, Manu, & Tommi. (2024). *Containe*rization in Multi*Cloud* Environment Roles Strategies Challenges and Solutions for Effective Implementation. https://arxiv.org/abs/2403.12980

André, Holger, Fei, Lucy, Claus, Stefan, & Johannes. (n.d.). 223226. https://dl.acm.org/doi/abs/10.1145/3053600.3053653

Annunen. (n.d.). Finding a suitable performance testing tool. https://oulurepo.oulu.fi/handle/10024/18348

David. (n.d.). Continuous delivery reliable software releases through build test and *Deployment* automation. Pearson Education. https://books.google.com/books?hl=en&lr=&id=6ADDuzere-YC&oi=fnd&pg=PT34&dq=By+running+various+load+tests,+developers+can+determine+the+system%27s+capacity+and+performance+limits,+allowing+them+to+make+necessary+adjustments+to+ensure&ots=-xrqLRHaid&sig=UpK_vQfWDVQeixSLyQ1YNbSjPtQ

Janach. (n.d.). University of Rijeka. https://zir.nsk.hr/islandora/object/infri:1316

Koziolek. (2010). Performance evaluation 67 no. https://www.sciencedirect.com/science/article/pii/S016653160900100X

Peng, Mojtaba, Amleto, & Gastón. (2021). Design monitoring and testing of *microservices* systems The practitioners perspective. https://www.sciencedirect.com/science/article/pii/S0164121221001588

Proskurin. (n.d.). Adapting a Stress Testing Framework to a Multi-module Security-oriented Spring Application. https://core.ac.uk/download/pdf/237084834.pdf

Rezvan, & Laurie. (2019). A systematic mapping study of infrastructure as code research. https://www.sciencedirect.com/science/article/pii/S0950584918302507

S. (2020). Comprehensive review of load testing tools. https://www.academia.edu/download/64573407/IRJET-V7I5651.pdf

Schumann. (n.d.). Conceptual design of a *container*based system landscape orchestrated by *Kubernetes*. https://libdoc.fh-zwickau.de/opus4/files/17421/BachelorThesisMaximilianSchumann.pdf

Sharmila, & Pradeep. (2023). Journal of Science Technology 4 no. https://www.thesciencebrigade.com/jst/article/view/340

Thakur. (n.d.). *Contain*erization in Web Development Streamlining *Deployment* and *Networ*king with Docker and *Kubernetes*. https://insights2techinfo.com/*contain*erization-in-web-development-streamlining-*Deployment*-and-*networ*king-with-docker-and-*Kuber*netes/

Xinjie, Tianjing, Guangwei, & Baek-Yong. (2018). Application *Deployment* using *Microservice* and Docker *containe*rs Framework and optimization. https://www.sciencedirect.com/science/article/pii/S1084804518302273

Yogesh. (n.d.). A pragmatic evaluation of stress and performance testing technologies for web based applications. https://ieeexplore.ieee.org/abstract/document/8701327/

Aakash, Peng, Muhammad, Arif, Iftikhar, Manu, & Tommi. (2024). *Contain*erization in Multi*Cloud* Environment Roles Strategies Challenges and Solutions for Effective Implementation. https://arxiv.org/abs/2403.12980

André, Holger, Fei, Lucy, Claus, Stefan, & Johannes. (n.d.). 223226. https://dl.acm.org/doi/abs/10.1145/3053600.3053653

Annunen. (n.d.). Finding a suitable performance testing tool. https://oulurepo.oulu.fi/handle/10024/18348

David. (n.d.). Continuous delivery reliable software releases through build test and *Deployment*

automation. Pearson Education. https://books.google.com/books?hl=en&lr=&id=6ADDuzere-YC&oi=fnd&pg=PT34&dq=By+running+various+load+tests,+developers+can+determine+the+system%27s+capacity+and+performance+limits,+allowing+them+to+make+necessary+adjustments+to+ensure&ots=-xrqLRHaid&sig=UpK_vQfWDVQeixSLyQ1YNbSjPtQ

Janach. (n.d.). University of Rijeka. https://zir.nsk.hr/islandora/object/infri:1316

Koziolek. (2010). Performance evaluation 67 no. https://www.sciencedirect.com/science/article/pii/S016653160900100X

Peng, Mojtaba, Amleto, & Gastón. (2021). Design monitoring and testing of *microservices* systems The practitioners perspective. https://www.sciencedirect.com/science/article/pii/S0164121221001588

Proskurin. (n.d.). Adapting a Stress Testing Framework to a Multi-module Security-oriented Spring Application. https://core.ac.uk/download/pdf/237084834.pdf

QuillBot. (2024). QuillBot Flow. (Dec 2024 *Version*) [Large Language Model]. Retrieved December 1, 2024, from https://quillbot.com/flow

Rezvan, & Laurie. (2019). A systematic mapping study of infrastructure as code research. https://www.sciencedirect.com/science/article/pii/S0950584918302507

S. (2020). Comprehensive review of load testing tools. https://www.academia.edu/download/64573407/IRJET-V7I5651.pdf

Schumann. (n.d.). Conceptual design of a *container*based system landscape orchestrated by *Kubernetes*. https://libdoc.fh-zwickau.de/opus4/files/17421/BachelorThesisMaximilianSchumann.pdf

Sharmila, & Pradeep. (2023). Journal of Science Technology 4 no. https://www.thesciencebrigade.com/jst/article/view/340

Thakur. (n.d.). *Containe*rization in Web Development Streamlining *Deployment* and *Networ*king with Docker and *Kubernetes*. https://insights2techinfo.com/*containe*rization-in-web-development-streamlining-*Deployment*-and-*networ*king-with-docker-and-*Kubernetes*/

Xinjie, Tianjing, Guangwei, & Baek-Yong. (2018). Application *Deployment* using *Microservice* and Docker *containe*rs Framework and optimization. https://www.sciencedirect.com/science/article/pii/S1084804518302273

Yogesh. (n.d.). A pragmatic evaluation of stress and performance testing technologies for web based applications. https://ieeexplore.ieee.org/abstract/document/8701327/

Aakash, Peng, Muhammad, Arif, Iftikhar, Manu, & Tommi. (2024). *Containe*rization in Multi*Cloud* Environment Roles Strategies Challenges and Solutions for Effective Implementation. https://arxiv.org/abs/2403.12980

Butter. (n.d.). PhD diss. https://epub.technikum-wien.at/obvftwhsm/content/titleinfo/10097580/full.pdf

Jasmin, Matej, & Luka. (2024). Computers 13 no. https://www.mdpi.com/2073-

431X/13/11/283

Jasmin, Matej, & Luka. (2024). Electronics 13 no. https://www.mdpi.com/2079-9292/13/19/3972

Nafise. (n.d.). Lightweight *Kubernetes* distributions A performance comparison of microk8s k3s k0s and microshift. https://dl.acm.org/doi/abs/10.1145/3578244.3583737

Pokkuluri. (2024). Requirements Engineering for Maintainability in *Microservices* using Trace Statistical Data From Literature to Experimental Simulation. https://www.diva-portal.org/smash/record.jsf?pid=diva2:1883352

Rafael, José, Unai, & Omer. (2018). Characterising resource management performance in *Kubernetes*. https://www.sciencedirect.com/science/article/pii/S0045790617315240

Shazibul, Jonathan, Patrick, & Akond. (2022). Benefits Challenges and Research Topics A Multivocal Literature Review of *Kubernetes*. https://arxiv.org/abs/2211.07032

Sotiris, Ioanna, Lefteris, & Vassilis. (2024). Performance evaluation of *Kubernetes networ*king approaches across constraint edge environments. https://arxiv.org/abs/2401.07674

Hafidz. (n.d.). PhD diss. https://repository.nurulfikri.ac.id/id/eprint/528/

Muhammad, & Chandra. (2024).... Monitoring Data Center Infrastructure Management (Dcim) Yang Terintegrasi Grafana Alert Dan Bot Telegram Dengan *Arsitektur Kubernetes* Studi Kasus Pt.... https://openlibrarypublications.telkomuniversity.ac.id/index.php/appliedscience/article/view/23812

Salma, & Muhammad. (2023). Monitoring *Kubernetes Cluster* Menggunakan Prometheus dan Grafana. https://abecindonesia.org/proceeding/index.php/abec/article/download/309/306

Aakash, Peng, Muhammad, Arif, Iftikhar, Manu, & Tommi. (2024). *Containe*rization in Multi*Cloud* Environment Roles Strategies Challenges and Solutions for Effective Implementation. https://arxiv.org/abs/2403.12980

Butter. (n.d.). PhD diss. https://epub.technikum-wien.at/obvftwhsm/content/titleinfo/10097580/full.pdf

Jasmin, Matej, & Luka. (2024). Computers 13 no. https://www.mdpi.com/2073-431X/13/11/283

Jasmin, Matej, & Luka. (2024). Electronics 13 no. https://www.mdpi.com/2079-9292/13/19/3972

Nafise. (n.d.). Lightweight *Kubernetes* distributions A performance comparison of microk8s k3s k0s and microshift. https://dl.acm.org/doi/abs/10.1145/3578244.3583737

Pokkuluri. (2024). Requirements Engineering for Maintainability in *Microservices* using Trace Statistical Data From Literature to Experimental Simulation. https://www.diva-portal.org/smash/record.jsf?pid=diva2:1883352

Rafael, José, Unai, & Omer. (2018). Characterising resource management performance in *Kubernetes*. https://www.sciencedirect.com/science/article/pii/S0045790617315240

Shazibul, Jonathan, Patrick, & Akond. (2022). Benefits Challenges and Research Topics A Multivocal Literature Review of *Kubernetes*. https://arxiv.org/abs/2211.07032

Sotiris, Ioanna, Lefteris, & Vassilis. (2024). Performance evaluation of *Kubernetes networ*king

approaches across constraint edge environments. https://arxiv.org/abs/2401.07674

Hafidz. (n.d.). PhD diss. https://repository.nurulfikri.ac.id/id/eprint/528/

Muhammad, & Chandra. (2024).... Monitoring Data Center Infrastructure Management (Dcim) Yang Terintegrasi Grafana Alert Dan Bot Telegram Dengan *Arsitektur Kubernetes* Studi Kasus Pt.... https://openlibrarypublications.telkomuniversity.ac.id/index.php/appliedscience/article/view/23812

Salma, & Muhammad. (2023). Monitoring *Kubernetes Cluster* Menggunakan Prometheus dan Grafana. https://abecindonesia.org/proceeding/index.php/abec/article/download/309/306

Athanasios, Dimitrios, Vasileios, Evangelos, Thomas, G., & Panagiotis. (n.d.). *Kubernetes in Edge and Cloud Computing: A Comparative Study of K3s, K0s, MicroK8s, and K8s*. https://www.researchgate.net/profile/Georgios-Papadopoulos-2/publication/389465530_Kubernetes_in_Edge_and_Cloud_Computing_A_Comparative_Study_of_K3s_K0s_MicroK8s_and_K8s/links/67c316528311ce680c791c59/Kubernetes-in-Edge-and-Cloud-Computing-A-Comparative-Study-of-K3s-K0s-MicroK8s-and-K8s.pdf

Bediatra. (n.d.). *Pengembangan Dashboard untuk Monitoring Celah Keamanan pada Cluster Kubernetes menggunakan Metode Container Images Scanning*. https://dspace.uii.ac.id/handle/123456789/47956

Berkah, Guson, Irwan, & Yusuf. (2025). *Tinjauan Pustaka Kubernetes Container Management*. https://repository.bakrie.ac.id/id/eprint/11162

Dani, & Agus. (2024). *Analisis Perbandingan Kinerja High Availability pada Cluster Docker Swarm dan K3s*. https://ejournal.unesa.ac.id/index.php/jinacs/article/view/60626

Ernawati, & Funny. (2021). *Implementasi Arsitektur Microservices Pada Rancang Bangun Aplikasi Marketplace Berbasis Web (Studi Kasus: Pasar Tradisional Modern Kota Bengkulu . . .*.

Hafidz. (n.d.). *. . . /CONTINUOUS DEPLOYMENT (CI/CD) DALAM PENGEMBANGAN WEB APLIKASI DI LINGKUNGAN KUBERNETES BERBASIS RANCHER KUBERNETES ENGINE*. https://repository.nurulfikri.ac.id/id/eprint/528/

I. (2023). *Analisis Perbandingan High Availibility Pada Web Server Menggunakan Orchestration Tool Kubernetes Dan Docker Swarm*. https://ejournal.unesa.ac.id/index.php/jinacs/article/view/56039

Luís, Karima, & David. (n.d.). *Assessing kubernetes distributions: A comparative study*. https://ieeexplore.ieee.org/abstract/document/10608706/

Muhammad, Widhi, & Achmad. (2025). *Pengembangan IoT Gateway Terdistribusi Berbasis Kubernetes*. https://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/14484

Sofyan, Arief, Irsyad, & Bagas. (2024). *Analisa Performansi Ujian Online Berbasis Website Dengan Arsitektur Lightweight Kubernetes (K3S)*. https://www.jurnal.polgan.ac.id/index.php/jmp/article/view/14240