SMATIKA : STIKI Informatika Jurnal
Vol. 15, No. 1, Juni 2025, pp. 193~202
ISSN: 2087-0256, e-ISSN: 2580-6939

Analisis *Build Time Continuous Integration* pada Multi Node Jenkins Menggunakan Metode *Cache*

Febriyan Adji Saputro[1]
Muhammad Agung Nugroho[2*]
Eka Sahputra[3]
Alon Jala Tirta Segara[4]

[1]Program Studi Informatika, Universitas Teknologi Digital Indonesia, Jl. Raya Janti Jl. Majapahit No.143, Jaranan, Banguntapan, Kec. Banguntapan, Kabupaten Bantul, Daerah Istimewa Yogyakarta 55198, Indonesia
[2]Program Studi Teknik Informatika, Universitas Telkom, Kampus Purwokerto, Jl. DI Panjaitan No.128, Karangreja, Purwokerto Kidul, Kec. Purwokerto Sel., Kabupaten Banyumas, Jawa Tengah 53147, Indonesia
[3]Program Studi Sains Data, Universitas Telkom, Kampus Purwokerto, Jl. DI Panjaitan No.128, Karangreja, Purwokerto Kidul, Kec. Purwokerto Sel., Kabupaten Banyumas, Jawa Tengah 53147, Indonesia
[4]Program Studi Rekayasa Perangkat Lunak, Universitas Telkom, Kampus Purwokerto, Jl. DI Panjaitan No.128, Karangreja, Purwokerto Kidul, Kec. Purwokerto Sel., Kabupaten Banyumas, Jawa Tengah 53147, Indonesia
[1]febriyan.adji@students.utdi.ac.id, [2]magungnugroho@telkomuniversity.ac.id, [3]ekasahputra@telkomuniversity.ac.id, [4]alonhs@telkomuniversity.ac.id

***Penulis Korespondensi:**
Muhammad Agung Nugroho
magungnugroho@telkomuniversity.ac.id

Abstrak

Dalam era pengembangan perangkat lunak yang modern, kecepatan dan efisiensi proses build aplikasi menjadi faktor penting untuk meningkatkan produktivitas tim developer. Jenkins sebagai alat CI/CD yang populer dapat dimanfaatkan untuk menjalankan proses build secara terdistribusi di beberapa server. Namun, build time yang tinggi tetap menjadi tantangan, terutama untuk aplikasi berbasis container yang memerlukan banyak dependensi dan tahap kompilasi. Penelitian ini bertujuan agar durasi proses build image perangkat lunak berbasis container pada multi-node Jenkins dapat berjalan lebih cepat dengan memanfaatkan metode cache. Dalam penelitian ini dilakukan eksperimen dengan membandingkan build time aplikasi berbasis container sebelum dan sesudah penerapan metode caching. Hasil penelitian ini menunjukkan bahwa penggunaan metode cache pada aplikasi berbasis container dapat meningkatkan kecepatan build time hingga 43 - 76%. Hal ini menunjukkan bahwa metode cache sangat efektif dalam meningkatkan efisiensi build time pada arsitektur multi-node Jenkins. Dalam eksperiment penelitian ini diperoleh kesimpulan bahwa cache dalam proses build image aplikasi berbasis container dengan menggunakan docker image registry sebagai tempat penyimpanan build cache dapat meningkatkan kecepatan build time hingga 43 - 76% dibandingkan jika tanpa menggunakan cache.

Kata Kunci: Cache; Integrasi Berkelanjutan; Penyebaran Berkelanjutan; Docker; Jenkins

Abstract

In the modern era of software development, the speed and efficiency of the application build process are crucial factors in increasing the productivity of developer teams. Jenkins, a popular CI/CD tool, can be utilized to run distributed build processes across multiple servers. However, high build times remain a challenge, especially for container-based applications that require numerous dependencies and compilation stages. This study aims to accelerate the duration of the container-based software image build process on multi-node Jenkins by utilizing the cache method. In this study, an experiment was conducted comparing the build time of container-based applications before and after the implementation of the caching method. The results of this study indicate that the use of the cache method in container-based applications can increase build time speeds by 43-76%. This indicates that the cache method is very effective in increasing build time efficiency in the multi-node Jenkins architecture. In this research experiment, it was concluded that caching in the image build process of container-based applications using the Docker image registry as a storage place for the build cache can increase build time speeds by 43-76% compared to without using the cache.

Keywords: Cache; Continuous Deployment; Continuous Integration; Docker; Jenkins.

Cite: Saputro, F.A, et al., (2025). Analisis Build Time Continuous Integration pada Multi Node Jenkins Menggunakan Metode Cache. SMATIKA : STIKI Informatika Jurnal,15(1). doi: https://doi.org/10.32664/smatika.v15i01.2034

1. Pendahuluan

Continuous Integration dan *Continuous Deployment* (CI/CD) merupakan serangkaian praktik pengembangan perangkat lunak yang akan membantu pengembang untuk merilis perangkat lunak dengan handal, aman, dan efisien[1],[2]. Biasanya kedua praktik ini dijalankan secara berurutan, dimulai dari *Continuous Integration* kemudian jika berhasil maka akan dilanjutkan dengan proses *Continuous Deployment*. *Continuous Integration* biasanya terdiri atas praktik testing dan *build* perangkat lunak secara otomatis setelah repositori perangkat lunak terhubung dengan server CI [3], [4]. Berbeda dengan proses pengembangan aplikasi berbasis waterfall [5], CI/CD memiliki proses *Continuous Deployment* yang dapat melakukan serangkaian proses perilisan perangkat lunak ke lingkungan produksi secara otomatis[6].

Untuk membantu pengembang dalam menerapkan praktik CI/CD, dibutuhkan *tools* CI/CD seperti *Gitlab CI*, *Jenkins*, *Travis CI*, *AWS CodePipeline*, *Go CD*, dan *Circle CI* [7]. *Jenkins* merupakan salah satu jenis tool CI/CD yang cukup populer untuk saat ini dikarenakan dukungan komunitas yang cukup besar dan jumlah *plugin* yang cukup banyak [8],[9]. Sebagaimana tool CI/CD pada umumnya, *Jenkins* juga mendukung arsitektur *multi-node*, yaitu arsitektur yang memungkinkan *Jenkins* memiliki lebih dari satu mesin pengeksekusi tugas-tugas atau *jobs* CI/CD, sehingga tugas-tugas tersebut nantinya tidak hanya dieksekusi bertumpuk di satu *node Jenkins* saja melainkan bisa terdistribusi secara acak ke node-node *Jenkins* yang lain [10].

Dalam arsitektur *multi-node Jenkins*, setidaknya akan terdapat node yang bertindak sebagai *master* dan *node* yang bertindak sebagai *slave* [11]. Node Jenkins yang bertindak sebagai *master* bertugas untuk mendistribusikan job-job CI/CD ke node-node yang bertindak sebagai *slave*, dan node slave akan bertugas untuk mengeksekusi job-job CI/CD tersebut [12]. Secara umum, terdapat 2 jenis perangkat lunak jika ditinjau dari sisi pengemasan aplikasi pada saat proses produksi. Jenis pertama adalah perangkat lunak yang berbasis *container*, sedangkan jenis kedua adalah perangkat lunak yang tidak berbasis *container* [13]. Banyak sekali manfaat dari perangkat lunak yang dikemas dalam bentuk *container*, salah satunya adalah adanya jaminan konsistensi kondisi lingkungan tempat perangkat lunak dijalankan, baik perangkat lunak tersebut dijalankan pada saat pengembangan, testing, hingga produksi [14]. Salah satu jenis platform yang dapat digunakan untuk membangun hingga mengelola *container* adalah *Docker*. *Jenkins* mendukung implementasi CI/CD untuk perangkat lunak berbasis *container*, dikarenakan *Jenkins* dapat terintegrasi dengan *Docker* dengan sangat baik.

Pada *Continuous Integration* perangkat lunak berbasis *container*, biasanya terdapat proses *build image* yang fungsinya adalah untuk mengemas perangkat lunak tersebut menjadi sebuah *docker image*[15]. Durasi waktu yang dibutuhkan untuk melakukan proses *build image* suatu perangkat lunak biasanya tergantung dari seberapa banyak *dependency* yang dibutuhkan oleh perangkat lunak tersebut. Artinya, semakin banyak *dependency* yang dibutuhkan maka semakin lama juga proses build image yang akan dilakukan. *Docker* sendiri memiliki fitur *image layer caching* yang dapat dimanfaatkan untuk mengoptimasi durasi waktu proses *build image* suatu perangkat lunak [16]. Namun secara default fitur ini hanya akan berjalan dengan baik pada suatu perangkat lunak jika sebelumnya perangkat lunak ini sudah pernah dilakukan proses *build image* di node yang sama. Sehingga apabila suatu perangkat lunak belum pernah sama sekali dilakukan proses *build image* di suatu node Jenkins, maka fitur *image layer caching* untuk perangkat lunak ini tidak akan dapat berjalan di *node Jenkins* tersebut walaupun sebenarnya perangkat lunak ini sudah pernah dilakukan proses *build image* di node-node yang lain. Sehingga dapat disimpulkan bahwa fitur *docker image layer caching* bawaan tidak akan dapat berjalan secara efektif pada Jenkins yang sudah menerapkan arsitektur *multi-node* dikarenakan bisa jadi job *build image* suatu perangkat lunak dieksekusi oleh *node* yang berbeda dengan node yang sebelumnya pernah melakukan *build image* perangkat lunak tersebut[17]. Oleh karena itu, akan dilakukan penelitian tentang proses optimasi durasi waktu build image pada continuous integration terhadap perangkat lunak berbasis *container* menggunakan *Jenkins* dengan arsitektur multi-node. Penelitian ini

memanfaatkan fitur *docker image layer caching* dengan melakukan penambahan kostumisasi dari sisi perintah dan infrastruktur.

Penelitian terkait CI/CD menggunakan docker dan jenkins dalam pengembangan web [18] menghasilkan penyelesaian proses deployment selama 1 menit 58 detik dengan tingkat keberhasilan sebesar 90%. Selain itu, ujicoba CI/CD juga dilakukan dalam pengembangan aplikasi *learning management system* (LMS) di PT. Millenia Solusi Informatika [19] dengan menggunakan *gitlab*, *enzyme*, dan *docker* yang melakukan proses otomasi pada model pengembangan layanan LMS dan dapat mengurangi tikat kesalahan yang terjadi karena *human error*. Dalam pengujian implementasi CI/CD pada aplikasi myITS dengan model *single sign on* diperlukan waktu rata-rata pipeline CI/CD berjalan sekita 5,93 menit dengan memanfaatkan teknologi *Docker*, *Kubernetes*, *Sonarqube*, dan *jenkins* [20]. Peningkatan proses pileline CI/CD pada *jenkins* dapat dioptimalisasi dengan menggunakan algoritma *Grey Wolf Optimizer* yang dalam eksperimen ini [21] dapat meningkatkan kecepatan deployment aplikasi sebesar 41,2 %. Dari penelitian-penelitian diatas proses produksi perangkat lunak menggunakan CI/CD seperti *Jenkins*, *Gitlab*, *ArgoCD*, *SonarQube* [22] dapat mempersingkat waktu proses, meningkatkan kinerja, dan meningkatkan ketelitian dalam penemuan *bugs* pada aplikasi.

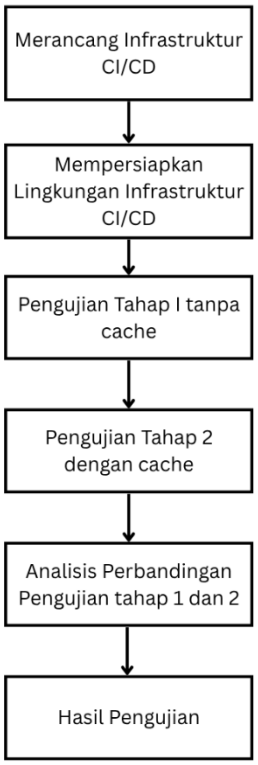
Penelitian ini dibatasi pada proses *build image* yang merupakan bagian dari pipeline CI di Jenkins. Pengujian dilakukan terhadap tiga aplikasi berbasis container yang dikembangkan menggunakan bahasa pemrograman *Node.js*, *Python*, dan *Ruby*. Masing-masing aplikasi dirancang memiliki minimal 20 *dependency* guna mensimulasikan waktu build yang signifikan. Penelitian ini akan membandingkan hasil pengujian sebelum dan sesudah penerapan optimasi *caching*, untuk mengukur efektivitas metode yang diusulkan. Proses cache secara konsep dapat meningkatkan proses build time pada building proses container. Dari eksperimen yang akan dilakukan pada penelitian ini dapat memberikan kontribusi berupa metode atau model cache untuk pengembangan aplikasi yang menerapkan model infrastruktur CI/CD yang menggunakan Jenkins dengan arsitektur *multi-node* dalam pengelolaan aplikasi berbasis *container*.

2. Metode Penelitian

Penelitian ini akan melakukan model percobaan untuk menguji model infrastruktur CI/CD dengan menggunakan *cache* dan tanpa *cache* untuk mengetahui proses peningkatan secara performa dalam proses infrastruktur melakukan *building image*. Gambaran penelitian ini dapat dilihat pada ilustrasi gambar 1.

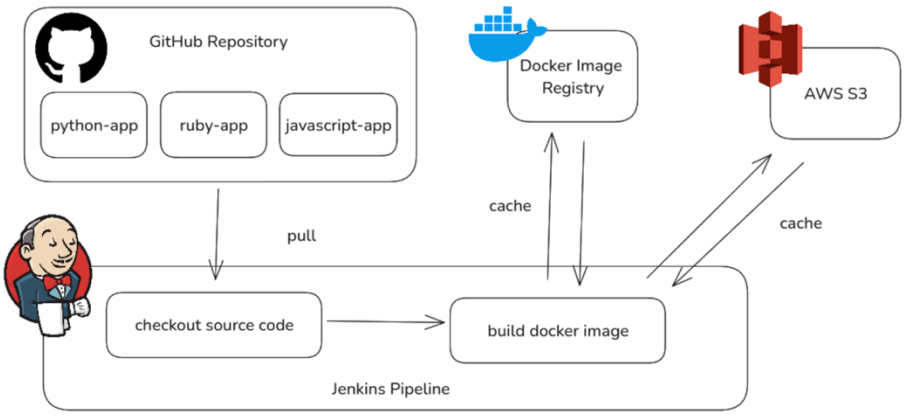
Untuk melakukan percobaan tahapan awal yang dilakukan adalah membangun sebuah infrastruktur dengan memanfaatkan layanan cloud. Rancangan infrastruktur *Continuous Integration Jenkins* pada penelitian ini bertujuan untuk mengoptimasi durasi waktu yang dihabiskan untuk melakukan proses *build image* perangkat lunak yang berbasis *container*. Infrastruktur *Continuous Integration* pada penelitian ini terbagi menjadi beberapa bagian :

1. *Stage checkout source code*, pada tahapan ini setelah tim developer melakukan push ke repositori *Github*, *jenkins* melakukan proses unduh *source code* dari repositori *Github* ke server *jenkins*.
2. *Stage Create Buildx Instance*, pada tahapan ini server jenkins menginisiasi *instance docker buildx* yang nantinya digunakan untuk melakukan *build docker image*.
3. *Stage build image*, pada tahapan ini server *jenkins* melakukan proses *build image* untuk mengubah *source code* perangkat lunak menjadi sebuah *docker image*. Durasi waktu yang dihabiskan pada tahapan ini menjadi tolak ukur utama penelitian ini.



Gambar 1. *Proses penelitian*

Seluruh layanan AWS (Amazon Web Service) yang digunakan pada penelitian ini terletak pada *region* dan *availability zone* yang sama yaitu *SouthEast 1 Zone A* agar proses komunikasi antar layanan menjadi lebih cepat dan optimal.



Gambar 2. *Rancangan Infrastruktur*

Fokus penelitian ini pada proses *build image* beberapa perangkat lunak dan tidak berhubungan dengan logika bisnis masing-masing perangkat lunak yang dioptimasi, maka beberapa perangkat lunak yang digunakan sebagai bahan pengujian merupakan beberapa perangkat lunak sederhana yang tidak memiliki banyak *source code*, namun memiliki jumlah *dependency/library* yang cukup banyak, yaitu berjumlah 20 buah atau lebih. Dengan adanya jumlah *dependency/library* yang cukup banyak membuat proses *build image* masing-masing perangkat lunak menjadi lebih lama sehingga perbedaan durasi waktu sebelum dan sesudah optimasi proses *build image* akan menjadi lebih mudah terlihat. 20 buah *dependency* yang digunakan dalam setiap perangkat lunak

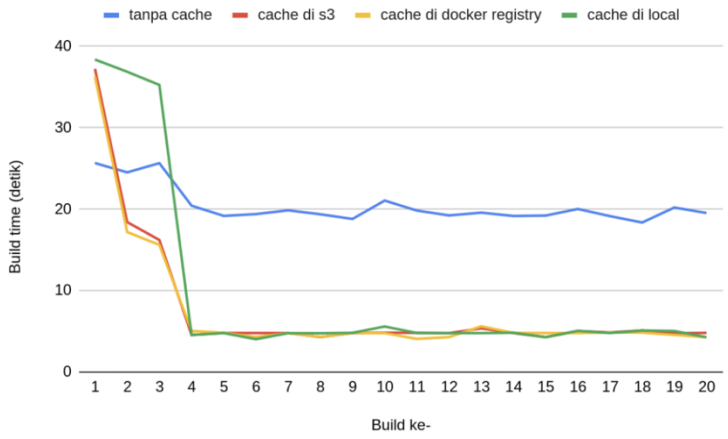
merupakan *dependency* acak yang diambil dari *dependencies repository* masing-masing *package manager* bahasa pemrograman.

Dalam pembangunan infrastruktur Continuous Integration yang teroptimasi proses *build image* dengan menggunakan metode *cache*, sehingga penelitian ini membutuhkan beberapa perangkat lunak untuk menjadi bahan pengujiannya. Penelitian ini menggunakan 3 buah perangkat lunak yang sudah berbasis *container* dengan bahasa pemrograman yang berbeda-beda yaitu *Javascript (Node JS)*, *Python*, *Ruby*. Masing-masing perangkat lunak tersebut merupakan perangkat lunak buatan penulis penelitian ini sendiri.

Proses pengujian pada gambar 1 dilakukan pada infrastruktur yang sudah direncanakan, dalam proses pengujian ini akan dilakukan dalam dua tahapan. Metriks yang digunakan pada penelitian ini adalah jumlah proses build, waktu rata-rata yang digunakan untuk build pada pipeline CI/CD dalam detik, dan pertambahan kecepatan build setelah menggunakan cache. Pada tahapan pertama adalah identifikasi masalah, yaitu dengan cara melakukan pengujian *build image* 3 buah aplikasi pada *multi-node jenkins* dengan tiap *pipeline* aplikasi dijalankan beberapa kali tanpa menggunakan metode *cache*. Masing-masing *pipeline* aplikasi dijalankan sebanyak 20 kali dengan melakukan perubahan *source code* tiap kali *pipeline* dijalankan, kemudian akan terlihat durasi waktu yang dihabiskan oleh masing-masing *pipeline* aplikasi tersebut dalam melakukan proses *build image* dengan keadaan belum teroptimasi. Angka durasi waktu dalam satuan detik yang terlihat pada *Jenkins* pada saat proses *build image* inilah yang menjadi tolak ukur utama penelitian ini. Selanjutnya, tahapan kedua adalah pengajuan solusi dari masalah yang sudah diidentifikasi yaitu dengan mengimplementasikan metode *cache* pada *pipeline* dengan harapan durasi waktu rata-rata yang dihabiskan untuk melakukan 20 kali proses *build image* pada 3 buah perangkat lunak menjadi lebih optimal.

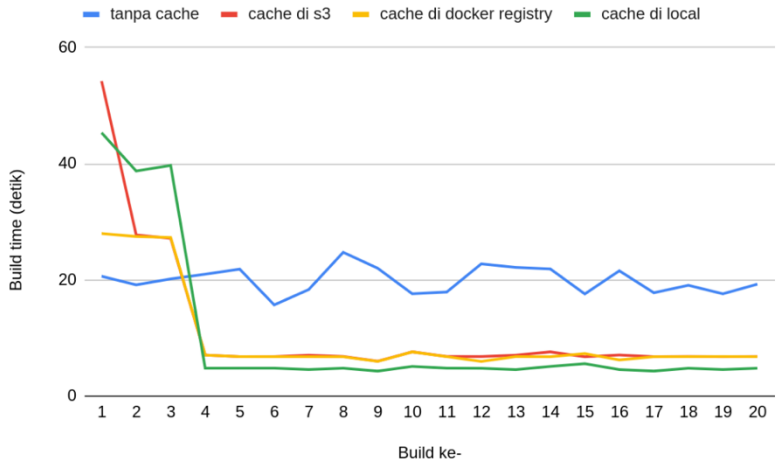
3. Hasil

Pengujian dilakukan terhadap 3 aplikasi dengan bahasa pemrograman yang berbeda, yaitu *python*, *javascript*, dan *ruby*. Pada hasil pengujian aplikasi *python* setelah dilakukan build sebanyak 20 kali dengan 4 metode yang berbeda menghasilkan data seperti pada Gambar 3 yang memperlihatkan perbandingan durasi waktu proses *build Docker image* dalam satuan detik terhadap 20 kali eksekusi pipeline pada Jenkins multi-node dengan empat skenario: tanpa *cache*, *cache* di S3, cache di *Docker registry*, dan *cache* di *local node*.



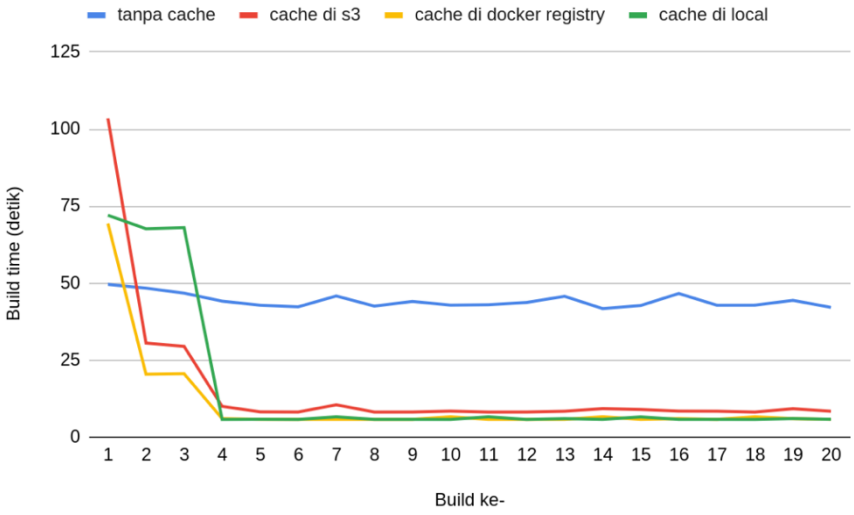
Gambar 3. *Pengujian build time aplikasi python*

Berdasarkan hasil pengujian pada aplikasi *python*, didapatkan bahwa menggunakan metode *cache* pada aplikasi *python* dapat menambah rata-rata kecepatan build time dibandingkan jika tidak menggunakan metode *cache* sama sekali. Kemudian dalam proses pengujian pada aplikasi *javascript* atau *nodejs* menunjukkan perbandingan waktu *build* (dalam detik) pada proses *Continuous Integration* menggunakan *Jenkins multi-node*, dengan empat skenario berbeda: tanpa *cache*, *cache* di S3, *cache* di *Docker registry*, dan *cache* di local node. Proses build sebanyak 20 kali dengan 4 metode yang berbeda menghasilkan data seperti pada gambar 4. Berdasarkan hasil pengujian pada aplikasi nodejs, didapatkan bahwa menggunakan metode *cache* pada aplikasi *nodejs* dapat menambah rata-rata kecepatan *build time* dibandingkan jika tidak menggunakan metode *cache* sama sekali.



Gambar 4. *Pengujian build time aplikasi nodejs*

Hasil pengujian dengan aplikasi *ruby* disajikan pada gambar 5 yang menunjukkan bahwa skenario tanpa *cache* menghasilkan waktu *build* yang stabil namun tinggi, berkisar antara 40 hingga 50 detik sepanjang 20 kali eksekusi sekitar 70 detik. Berdasarkan hasil pengujian pada aplikasi *ruby*, didapatkan bahwa menggunakan metode *cache* pada aplikasi *ruby* dapat menambah rata-rata kecepatan *build time* dibandingkan jika tidak menggunakan metode *cache* sama sekali.



Gambar 5. *Pengujian build time aplikasi ruby*

4. Pembahasan

Pengujian terhadap aplikasi *python* yang disajikan pada gambar 3 menunjukkan bahwa proses *build* tanpa *cache* memiliki waktu eksekusi yang paling tinggi dan relatif stabil, yaitu antara 19

hingga 26 detik. Hal ini terjadi karena setiap proses *build* melakukan eksekusi penuh tanpa memanfaatkan *cache layer* dari *build* sebelumnya. Sementara itu, penerapan *cache* pada semua metode menunjukkan penurunan waktu *build* yang sangat signifikan setelah beberapa eksekusi awal. *Cache* di S3 dan *Docker registry* menghasilkan waktu build yang rendah dan stabil mulai dari eksekusi kedua, mengindikasikan efektivitas metode *caching* eksternal dalam lingkungan *multi-node*. Sedangkan pada *cache* lokal, meskipun waktu *build* awal lebih tinggi, setelah *cache* tersimpan di node yang sama, waktu *build* menurun drastis. Jika dilihat dari lokasi *cache* maka detail penambahan kecepatannya pada proses *build pipeline* CI/CD terlihat pada tabel 1.

Tabel 1. *Hasil penambahan kecepatan built time pada python*

Lokasi *cache*	Rata-rata *built time* (detik)	Penambahan kecepatan
Lokal	9.538	53,18%
AWS S3	7.397	63,69%
Docker Registry	7.634	62,53%

Pengujian kedua dilakukan dengan menggunakan aplikasi javascript, dari hasil pengujian, skenario tanpa *cache* memiliki waktu *build* yang relatif konstan dan cukup tinggi, berada pada kisaran 18 hingga 25 detik. Sebaliknya, ketiga skenario caching menunjukkan pola penurunan waktu *build* secara signifikan setelah eksekusi awal. *Cache* lokal menunjukkan waktu *build* awal yang sangat tinggi, mendekati 45 detik, namun turun drastis menjadi di bawah 10 detik pada eksekusi keempat dan seterusnya. *Cache* di S3 dan *Docker registry* memiliki pola penurunan yang serupa, dengan waktu *build* awal sekitar 27–55 detik, lalu stabil di kisaran rendah, rata-rata di bawah 10 detik mulai eksekusi keempat. Penggunaan metode *caching* sangat efektif dalam mengurangi waktu *build image.* Metode *caching* eksternal seperti S3 dan *Docker registry* lebih konsisten pada lingkungan multi-node, sedangkan *cache* lokal efektif namun sensitif terhadap distribusi node. Jika dilihat dari lokasi *cache* maka detail penambahan kecepatannya pada proses *build pipeline* CI/CD terlihat pada tabel 2.

Tabel 2. *Hasil penambahan kecepatan built time pada nodejs*

Lokasi *cache*	Rata-rata *built time* (detik)	Penambahan kecepatan
Lokal	10.183	48,78%
AWS S3	9.826	50,57%
Docker Registry	11.264	62,53%

Penambahan kecepatan ini disebabkan tidak adanya *reuse layer image*, sehingga setiap proses *build* memerlukan kompilasi penuh. Sebaliknya, ketiga skenario *caching* memperlihatkan penurunan waktu *build* yang signifikan setelah beberapa eksekusi awal. Pada *cache* di S3 dan *Docker registry*, waktu *build* awal cukup tinggi, mencapai 100 detik pada eksekusi pertama, namun menurun drastis hingga stabil di bawah 10 detik mulai eksekusi keempat. *Cache* lokal juga mengalami penurunan serupa, meskipun memiliki waktu *build* awal Pemanfaatan caching sangat efektif dalam mengurangi waktu *build image*, dengan efisiensi pengurangan waktu mencapai lebih dari 80% pada *build* selanjutnya. Jika dilihat dari lokasi *cache* maka detail penambahan kecepatannya pada proses *build pipeline* CI/CD terlihat pada tabel 3.

Tabel 3. *Hasil penambahan kecepatan built time pada ruby*

Lokasi *cache*	Rata-rata *built time* (detik)	Penambahan kecepatan
Lokal	15.296	65,31%
AWS S3	10.447	76,31%
Docker Registry	15.397	65,08%

Berdasarkan hasil eksperimen yang dilakukan terhadap tiga aplikasi berbasis *container*, dapat disimpulkan bahwa penerapan metode *caching* pada proses *build image* dalam lingkungan *Jenkins multi-node* secara signifikan meningkatkan efisiensi waktu build. Penerapan metode *cache* menghasilkan peningkatan rata-rata kecepatan *build time* sebesar 43% hingga 76%, bila dibandingkan dengan proses *build* tanpa *cache*. Peningkatan ini menunjukkan bahwa pemanfaatan cache mampu mengurangi *overhead* proses kompilasi dan pengambilan dependensi berulang pada setiap eksekusi *pipeline*.

Dari hasil pengujian yang dilakukan menggunakan tiga jenis metode *cache*, yaitu *cache* lokal, *cache* berbasis Amazon S3, dan *cache* melalui *Docker image registry* menunjukkan peningkatan performa yang positif terhadap *build time*, namun dengan tingkat efektivitas yang bervariasi. Di antara ketiga metode tersebut, *cache* yang memanfaatkan *Docker image registry* menunjukkan hasil paling optimal dalam hal efisiensi waktu. Rata-rata peningkatan kecepatan *build time* yang diperoleh dari metode ini antara 43% hingga 65%. Hal ini dikarenakan *Docker image registry* memungkinkan distribusi *layer image* secara efisien antar *node Jenkins*, sehingga *cache* dapat diakses secara konsisten terlepas dari *node* eksekutor yang digunakan. Metode *cache* lokal, meskipun memberikan *build time* tercepat setelah *cache* tersedia, memiliki keterbatasan dalam konteks arsitektur *multi-node* karena *cache* tidak tersedia lintas *node*. Sebaliknya, metode *cache* berbasis S3 menawarkan performa mendekati *registry*, namun dengan latensi yang sedikit lebih tinggi karena proses pengambilan dan penyimpanan artefak memerlukan koneksi jaringan ke storage eksternal. Persamaan penelitian ini dengan penelitian-penelitian sebelumnya adalah sama-sama menggunakan Jenkins dan Docker dalam implementasi CI/CD nya, namun pustaka-pustaka di atas belum menerapkan metode cache dalam proses build image nya, sehingga durasi waktu untuk melakukan proses build image belum teroptimasi dengan maksimal. Proses pengujian dari penggunaan metode cache menunjukkan peningkatan kecepatan build time.

5. Penutup

Berdasarkan hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa pemanfaatan mekanisme *cache* dalam proses *build image* aplikasi berbasis container pada arsitektur multi-node *Jenkins* memberikan dampak signifikan terhadap efisiensi waktu proses *build*. Penerapan metode *cache* terbukti mampu mempercepat durasi *build image* secara konsisten dengan peningkatan performa sebesar 43% hingga 76% dibandingkan proses *build* tanpa cache.

Meskipun demikian, perlu dicatat bahwa efektivitas peningkatan *build time* tersebut tetap dipengaruhi oleh sejumlah faktor eksternal, seperti kecepatan jaringan antar-node, performa penyimpanan (disk I/O), jumlah dependency yang dibutuhkan oleh aplikasi, serta ukuran dan kompleksitas kode sumber yang dibangun.

Penelitian ini terbatas hanya meneliti cache pada level docker image layer dan proses akan semakin kompleks saat dependensi aplikasi semakin banyak yang akan mempengaruhi kecepatan build time. Selain itu, batasan penelitian ini hanya pada penggunaan tools otomasi jenkins dan belum diperbandingkan dengan tools lain seperti Gitlab CI, Tekton, ArgoCD. Penelitian ini dapat pula dikembangkan dengan memaksimalkan metode caching sampai pada level dependensi aplikasi.

6. Referensi

[1] R. Queiroz, T. Cruz, J. Mendes, P. Sousa, and P. Simões, "Container-based Virtualization for Real-time Industrial Systems—A Systematic Review," *ACM Comput Surv*, vol. 56, no. 3, pp. 1–38, Mar. 2024, doi: 10.1145/3617591.

[2] V. Ivanov, D. Krasnikhin, S. Litvinov, S. Masyagin, and G. Succi, "A Lean and Devops Approach to Teach Lean Software Development," 2019, pp. 196–204. doi: 10.1007/978-3-030-06019-0_15.

[3] S. Bello, "Optimizing CI/CD Code Efficiency Through Intelligent Caching and Dependency Management," Jul. 2022.

[4] E. Wolff, *A Practical Guide to Continuous Delivery*. Addison-Wesley Professional, 2017.

[5] R. F. Rizaldi, S. Busono, and A. S. Fitrani, "Sistem Informasi Inventaris Barang Di UPTD Puskesmas Kemlagi Menggunakan Metode Waterfall," *SMATIKA JURNAL*, vol. 14, no. 01, pp. 13–22, Jun. 2024, doi: 10.32664/smatika.v14i01.1128.

[6] A. Mahgoub, "From Monolith to Microservices: Building Scalable Applications with Kubernetes, CI/CD, and Chaos Engineering," 2025.

[7] Z. Huang, H. Fan, B. Tang, S. Wu, C. Yu, and H. Jin, "CBuild: Cluster-oriented Collaborative Image Building for Containers," *IEEE Transactions on Computers*, pp. 1–12, 2025, doi: 10.1109/TC.2025.3575912.

[8] P. Chintale, *DevOps Design Pattern: Implementing DevOps Best Practices for Secure and Reliable CI/CD Pipeline (English Edition)*. Bpb Publications, 2023. [Online]. Available: https://books.google.com/books?hl=id&lr=&id=GIXrEAAAQBAJ&oi=fnd&pg=PT25&dq=Best+Practices+in+Docker+Registry+Management+for+CI/CD&ots=vmpozlnJeh&sig=7q2B7ywxJw095BBwiTI-7daaQpw

[9] R. Leszko, *Continuous Delivery with Docker and Jenkins: Create Secure Applications by Building Complete CI/CD Pipelines*. Packt Publishing Ltd, 2022. [Online]. Available: https://books.google.com/books?hl=id&lr=&id=3V5qEAAAQBAJ&oi=fnd&pg=PP1&dq=Best+Practices+in+Docker+Registry+Management+for+CI/CD&ots=sdgq4Sa9yG&sig=Au7o_fFpRQBx_zefUF4esyNOJVo

[10] S. R. Dileepkumar and J. Mathew, "Transforming Software Development: Achieving Rapid Delivery, Quality, and Efficiency with Jenkins-Based CI/CD Pipelines," in *2023 Annual International Conference on Emerging Research Areas: International Conference on Intelligent Systems (AICERA/ICIS)*, IEEE, Nov. 2023, pp. 1–6. doi: 10.1109/AICERA/ICIS59538.2023.10420251.

[11] F. A. Saputro, "Optimasi Build Time Continuous Integration pada Multi Node Jenkins dengan Menggunakan Metode Cache," Universitas Teknologi Digital Indonesia, 2024. [Online]. Available: https://eprints.utdi.ac.id/10509/

[12] S. Amgothu, "An End-to-End CI/CD Pipeline Solution Using Jenkins and Kubernetes," *International Journal of Science and Research (IJSR)*, vol. 13, no. 8, pp. 1576–1578, Aug. 2024, doi: 10.21275/SR24826231120.

[13] M. Agung Nugroho, Y. Kusnanto, R. Murdiantoro Aji, and Rikie Kartadie, "Analisis QoS dalam implementasi documenso pada kubernetes," *INFOTECH : Jurnal Informatika & Teknologi*, vol. 5, no. 2, pp. 267–279, Dec. 2024, doi: 10.37373/infotech.v5i2.1448.

[14] M. A. N. Nugroho and R. M. Aji, *Kubernetes Cluster: Implementasi Infrastruktur Aplikasi Digital Signature Documenso*. Deepublish, 2024.

[15] M. A. Nugroho and R. Kartadie, "CLOUD STORAGE DENGAN TEKNOLOGI KUBERNETES UNTUK PLATFORM COLLABORATIVE RESEARCH," *JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)*, vol. 6, no. 1, pp. 74–81, May 2021, doi: 10.29100/jipi.v6i1.1908.

[16] Z. Huang, S. Wu, S. Jiang, and H. Jin, "FastBuild: Accelerating Docker Image Building for Efficient Development and Deployment of Container," in *2019 35th Symposium on Mass*

Storage Systems and Technologies (MSST), IEEE, May 2019, pp. 28–37. doi: 10.1109/MSST.2019.00-18.

[17] R. Leszko, *Continuous Delivery with Docker and Jenkins: Create secure applications by building complete CI/CD pipelines*. Packt Publishing Ltd, 2022. [Online]. Available: https://books.google.com/books?hl=id&lr=&id=3V5qEAAAQBAJ&oi=fnd&pg=PP1&dq=jenkins+docker&ots=sdgq4Sc3xI&sig=rxMIkTArbkiQ_3GUMPfojiZ4GwY

[18] A. Alpery and M. A. F. Ridha, "Implementasi CI/CD Dalam Pengembangan Aplikasi Web Menggunakan Docker dan Jenkins," in *9th Applied Business and Engineering Conference*, Politeknik Caltex Riau, 2021.

[19] A. P. Wahyu and I. Guna Noviantama, "IMPLEMENTASI CONTIONOUS INTEGRATION DAN CONTINOUS DEPLOYMENT PADA APLIKASI LEARNING MANAGEMENT SYSTEM DI PT. MILLENNIA SOLUSI INFORMATIKA," *Jurnal Ilmiah Teknologi Infomasi Terapan*, vol. 8, no. 1, pp. 183–186, Dec. 2021, doi: 10.33197/jitter.vol8.iss1.2021.744.

[20] R. A. Parama, H. Studiawan, and R. J. Akbar, "Implementasi Continuous Integration dan Continuous Delivery Pada Aplikasi myITS Single Sign On," *Jurnal Teknik ITS*, vol. 11, no. 3, Dec. 2022, doi: 10.12962/j23373539.v11i3.99436.

[21] R. Adrian, A. K. Fauziyyah, and S. Alam, "Continuous Integration/ Continuous Delivery Optimization on Network Automation using Gray Wolf Optimizer," *SISTEMASI*, vol. 11, no. 3, p. 776, Sep. 2022, doi: 10.32520/stmsi.v11i3.2322.

[22] A. D. Laksito, "Implementasi Continuous Integration/Continuous Delivery (Ci/Cd) Pada Performance Testing Devops," *Jurnal Informatika Sistem dan Manajemen (JOISM)*, vol. 4, no. 1, pp. 62–66, 2022.