Journal of Information Technology and Education (JITED)
Vol. 3 No. 1, March 2025 , Pages: 101-111, ISSN: 3026-6564 (Online)

IMPLEMENTASI CONTINUOUS INTEGRATION DAN CONTINUOUS DELIVERY (CI/CD) PADA MODEL DEEP LEARNING DENGAN GOOGLE CLOUD PLATFORM STUDI KASUS PEMBANGKIT SOAL OTOMATIS

Dhista Dwi Nur Ardiansyah[1], Handaru Jati[2]
Pendidikan Teknik Informatika, Universitas Negeri Yogyakarta
Email: [1] dhistadwi.2019@student.uny.ac.id, [2] handaru@uny.ac.id

ABSTRAK

Proses *deployment* yang tidak efektif dapat berdampak buruk pada perilisan aplikasi kepada pengguna dan stabilitas sistem, yang pada gilirannya akan mempengaruhi pengalaman pengguna dalam menggunakan aplikasi. Tujuan penelitian ini adalah mengimplementasikan penggunaan Docker dan Kubernetes pipeline melalui pendekatan DevOps MLOps untuk meningkatkan efektivitas proses *deployment*, dan menganalisis kinerja penggunaan Docker dan Kubernetes sebagai alternatif *environment* dalam melakukan *deployment* web server AQG. Metode yang digunakan adalah *Research and Development* dengan prosedur pengembangan *Development and Operations* yang terdiri dari *plan, code & build, test, release, deploy & operate,* monitor. Dengan memperhatikan waktu *deploy* dan kinerja web server dengan menggunakan Docker dan Kubernetes *Pipeline* yang didapat melalui analisis hasil *load testing* pada web server. Subjek dari penelitian ini adalah hasil kinerja web server yang menggunakan Docker dan Kubernetes *Pipeline*. Hasil penelitian ini mencakup (1) pengembangan web server untuk layanan model *Automatic Question Generator* dengan menggunakan Docker dan Kubernetes *Pipeline*, dan (2) evaluasi *load testing* menunjukkan bahwa web server dengan Docker dan Kubernetes *Pipeline* lebih stabil dalam hal *error rate*, *throughput*, dan *response time* saat *threads* meningkat dari 5.000 menjadi 10.000. Total waktu *deploy* juga mengalami penurunan dari 9 menit 1 detik menjadi 3 menit 40 detik, menunjukkan peningkatan efisiensi sebesar 245%.

Kata kunci: DevOps, Docker, Kubernetes, Cloud Computing

ABSTRACT

Ineffective deployment processes can negatively impact application releases to users, while unstable systems can undermine user experience. This research aims to implement Docker and Kubernetes Pipelines through the DevOps MLOps approach to enhance deployment process effectiveness and analyze the performance of Docker and Kubernetes as alternative environments for deploying the AQG web server. The research methodology follows Research and Development with Development and Operations procedures encompassing planning, coding & building, testing, releasing, deploying & operating, and monitoring. The study focuses on deploy time and web server performance using Docker and Kubernetes Pipelines, assessed through load testing analysis. The subject of this research is the performance outcomes of the web server utilizing Docker and Kubernetes Pipelines. The research outcomes include development of a web server for the Automatic Question Generator service using Docker and Kubernetes Pipelines. Load testing evaluations demonstrate that the web server with Docker and Kubernetes Pipelines exhibits greater stability in error rate, throughput, and response time as threads increase from 5,000 to 10,000. Moreover, the total deploy time decreased from 9 minutes 1 second to 3 minutes 40 seconds, indicating a 245% increase in efficiency.

***Keyword*:** DevOps, Docker, Kubernetes, Cloud Computing

PENDAHULUAN

Salah satu permasalahan dalam evaluasi pembelajaran adalah kesulitan guru dalam membuat banyak butir soal dalam waktu yang terbatas. Hal ini disebabkan oleh beban kerja yang cukup berat sebagai pendidik, termasuk melaksanakan administrasi dan pengelolaan pendidikan sesuai dengan Undang-Undang Republik Indonesia No. 20 Tahun 2023. Selain itu, guru juga kesulitan mengembangkan instrumen evaluasi pembelajaran, sehingga sering menggunakan soal tahun sebelumnya [1]. Pendekatan ini tidak efektif dan dapat mengarah pada kekurangan variasi dalam soal serta pembelajaran yang kurang memadai bagi siswa.

Untuk mengatasi tantangan ini, diimplementasikanlah sistem Automatic Question Generator (AQG) yang memanfaatkan teknologi kecerdasan buatan atau *artificial intelligence* (AI) dalam pembuatannya [2]. AQG adalah sebuah sistem yang dapat membuat soal dan jawaban secara otomatis, serta menyediakan manajemen soal secara efisien [3]. Namun, pengembangan model AI, terutama dalam cabang Natural Language Processing (NLP) yang digunakan dalam AQG, memerlukan proses *training* yang panjang karena kompleksitasnya yang tinggi [4]. Proses ini melibatkan pengumpulan dan pelabelan data, pemilihan dan penyesuaian model, serta pengujian yang teliti untuk memastikan kualitas dan keakuratan soal yang dihasilkan. Oleh karena itu, meskipun AQG menjanjikan solusi efisien untuk membuat soal evaluasi, proses pengembangan dan pelatihan model AI membutuhkan komitmen waktu dan sumber daya yang signifikan [5].

Setelah melalui tahap *training*, model AQG akan diimplementasikan dalam bentuk web server. Proses *deployment* web server sering kali dilakukan menggunakan cara manual, yang berpotensi menyebabkan inkonsistensi performa dan risiko tinggi karena tidak terintegrasi dengan baik [6]. Selain itu, perawatan model juga penting untuk mempertahankan kondisi optimalnya. Model *machine learning* perlu menjalani proses *maintenance* reguler, termasuk *training* ulang secara berkala, karena cenderung mengalami penurunan performa seiring waktu [7]. Proses *deployment* yang memakan waktu dan sering perlu diulang menyebabkan penggunaan waktu yang tidak efisien [6]. Dalam konteks pembaruan aplikasi, DevOps memiliki peranan krusial. Pendekatan ini mengintegrasikan pengembangan dan operasi untuk meningkatkan kolaborasi, mempercepat pengiriman perangkat lunak, dan meningkatkan efisiensi keseluruhan [8], [9]. Dengan menerapkan prinsip-prinsip DevOps, perusahaan dan pengembang aplikasi dapat melakukan pembaruan dan perilisan aplikasi secara teratur, cepat, dan stabil, sehingga tetap bersaing di pasar yang semakin ketat [10]. Dengan adanya DevOps, proses pengembangan dan perilisan aplikasi menjadi lebih terintegrasi, efisien, dan responsif terhadap perubahan pasar dan kebutuhan pengguna.

DevOps adalah metodologi yang dikenal dapat meningkatkan efektivitas dan efisiensi dalam pengembangan produk dengan mengotomatisasi tahap pengembangan dan operasi. Penelitian oleh menunjukkan bahwa penerapan DevOps dapat mempercepat pengembangan dan perilisan produk hingga 72%, mulai dari interval harian hingga enam bulan [11]. Studi lain oleh [12] memvalidasi manfaat DevOps, termasuk penyingkatan waktu pengembangan, penurunan tingkat kesalahan, pengurangan cacat selama pengembangan dan operasi, serta optimasi sumber data [10].

MLOps, sebagai perluasan dari DevOps ke dalam pengembangan sistem pembelajaran mesin, mengintegrasikan alat, *pipeline*, dan proses pembuatan model untuk menghemat waktu dan biaya [13]. Dalam konteks pengembangan Automatic Question Generator (AQG), yang penting untuk Sistem Tutor Cerdas (ITS) dalam fase pengetahuan domain [14], penggunaan CI/CD *Pipeline* dapat meningkatkan efisiensi dalam pengembangan, operasi, dan pemeliharaan. Siklus hidup MLOps diatur oleh *pipeline* CI/CD, mengotomatisasi proses untuk mempercepat perilisan [15].

Berdasarkan uraian di atas, masalah yang menjadi dasar penelitian adalah kurangnya efektivitas dalam proses *deployment* layanan model Automatic Question Generator (AQG) dalam bentuk web server ke lingkungan produksi, serta ketidakstabilan kinerja web server yang di-*deploy* menggunakan metode tradisional. Penelitian ini bertujuan untuk meningkatkan efektivitas proses pengembangan sistem AQG dan melakukan analisis terhadap kinerja CI/CD *Pipeline* sebagai alternatif lingkungan dalam melakukan proses *deployment*. Dengan demikian, penelitian ini akan menginvestigasi perbedaan efisiensi antara penggunaan CI/CD *Pipeline* dan metode tradisional dalam proses *deployment* serta dampaknya terhadap stabilitas kinerja web server AQG.

TUJUAN PENELITIAN

1. Mengimplementasikan penggunaan Docker dan Kubernetes *Pipeline* melalui metode DevOps pendekatan MLOps untuk meningkatkan efektivitas proses *deployment*
2. Menganalisis hasil kinerja penggunaan Docker dan Kubernetes sebagai alternatif *environment* dalam melakukan *deployment* web server AQG

METODE

Penelitian ini menggunakan metode *Research & Development* (R&D) dengan tujuan membuat CI/CD *Pipeline* untuk layanan web server model Automatic Question Generator (AQG). CI/CD *Pipeline* dibuat menggunakan tahapan DevOps: (1) *plan*, (2) *code*, (3) *build*, (4) *test*, (5) *release*, (6) *deploy*, (7) *operate*, dan (8) monitor. Web server yang dirilis menggunakan CI/CD *Pipeline* akan dibandingkan kinerjanya dengan web server yang dirilis dengan metode tradisional dengan mengukur kinerja web server melalui proses *load testing*. Dalam *load testing* akan dilakukan pengetesan

dengan 2 skenario yaitu dengan 5.000 pengguna virtual dan 10.000 pengguna virtual. *Load testing* akan dilakukan menggunakan Apache Jmeter.

PROSEDUR PENGEMBANGAN

Metode DevOps memiliki 8 tahapan, yaitu:

1. *Plan*
   Dalam tahap plan, dilakukan identifikasi kebutuhan untuk pembuatan CI/CD *Pipeline*, pemilihan model AQG yang memiliki akurasi tertinggi, serta pemilihan arsitektur layanan web server AQG yang praktis dan efisien. Proses pembuatan *pipeline* diperkuat dengan penggunaan Docker dan Kubernetes sebagai lingkungan (*environment*) untuk proses *deployment*. Tahapan ini menjadi krusial dalam memastikan kelancaran dan efektivitas implementasi CI/CD *Pipeline* serta kesesuaian model AQG dan arsitektur web server dengan tujuan aplikasi yang dikehendaki.
2. *Code*
   Proses *code* atau pembuatan web server akan dilakukan menggunakan bahasa pemrograman Python dan *framework* Flask. Pemilihan Python didasarkan pada kemudahan penggunaannya dan dukungan yang luas dalam pengembangan aplikasi web. Sementara itu, pemilihan Flask sebagai *framework* untuk pengembangan web server didasarkan pada keunggulannya yang ringan, fleksibel, dan memungkinkan pengembangan aplikasi web yang cepat dan efisien [16]. Kode web server yang telah dibuat akan disimpan pada GitHub sebagai repositori kode dan untuk pengaturan versi.
3. *Build*
   Proses *build* kode menggunakan Docker. Docker dipilih karena kemampuannya membangun kode dan dependensinya ke dalam kontainer terisolasi, yang memungkinkan portabilitas dan konsistensi di berbagai lingkungan pengembangan dan produksi [17]. Setelah kode berhasil dibangun menjadi kontainer, kontainer tersebut akan diunggah ke Docker Hub sebagai *registry* artefak. Docker Hub merupakan layanan *registry* yang populer untuk menyimpan, mengelola, dan berbagi kontainer Docker, sehingga mempermudah proses distribusi dan penyebaran aplikasi web server ke berbagai lingkungan pengembangan dan produksi.
4. *Test*
   Proses *test* merupakan tahap penting dalam memastikan bahwa web server beroperasi dengan kualitas dan kinerja optimal. *Unit test* menjadi fokus utama dalam menguji fungsionalitas dan integritas kode di dalam web server, dengan tujuan memverifikasi bahwa setiap komponen dan fungsi beroperasi sesuai harapan dan menghasilkan output yang akurat. Uji tes ini menjadi dasar yang kritis untuk memastikan bahwa web server berfungsi dengan

baik dan memenuhi kebutuhan pengguna.

5. *Release*
   Web server yang telah melalui proses pengujian akan dirilis untuk digunakan. Proses rilis ini dimaksudkan untuk memastikan bahwa web server telah siap digunakan di lingkungan produksi dan mampu memberikan layanan yang konsisten.
6. *Deploy*
   Tahapan *deploy* dan operasi web server akan dilaksanakan di Google Cloud Platform (GCP). Tujuan dari tahap *deploy* adalah untuk menyiapkan infrastruktur yang dibutuhkan agar web server dapat berjalan di lingkungan produksi. Setelah berhasil di-*deploy*, web server akan beroperasi di lingkungan GCP. Pada tahap ini, juga akan dipersiapkan skrip untuk melakukan *deployment* menggunakan GitHub Actions.
7. *Operate*
   Proses *operate* dilakukan melalui Google Kubernetes Engine dengan pembuatan *cluster* dan penugasan *workload* pada *cluster* yang telah dibuat. Tahapan *deploy* akan otomatis dilakukan dengan dukungan GitHub Actions. Selain tahap *deploy*, proses *build, test,* dan *release* juga akan otomatis dilaksanakan.
8. Monitor
   Web server terus dimonitor secara kontinu, termasuk performa, log aplikasi, dan kesalahan aplikasi. Monitoring web server dapat dilakukan melalui *dashboard monitoring* yang tersedia di konsol Google Cloud Platform.

WAKTU DAN TEMPAT PENELITIAN

Penelitian Implementasi Continuous Integration dan Continuous Delivery (CI/CD) pada Model Deep Learning dengan Google Cloud Platform Studi Kasus Pembangkit Soal Otomatis dilakukan di Universitas Negeri Yogyakarta, selama Mei 2023 hingga September 2023 .

DATA, INSTRUMEN, DAN TEKNIK PENGUMPULAN

Web server yang telah diterapkan baik melalui CI/CD Pipeline maupun menggunakan metode tradisional akan mengalami pengujian beban. Evaluasi beban pada kedua metode tersebut akan mencatat rata-rata waktu respons, tingkat kesalahan, dan *throughput*. Selain itu, perbandingan waktu yang diperlukan untuk melakukan *deploy* antara CI/CD *Pipeline* dan metode tradisional juga akan dicatat.

TEKNIK ANALISIS DATA

Teknik analisis data merujuk pada suatu metode untuk mengolah serta menganalisis data yang telah terkumpul, berdasarkan instrumen penelitian yang digunakan. Dalam konteks evaluasi hasil *load testing* web server menggunakan CI/CD Pipeline dan metode tradisional, analisis data dilakukan dengan membandingkan kinerja dari kedua metode tersebut. Tujuannya adalah untuk

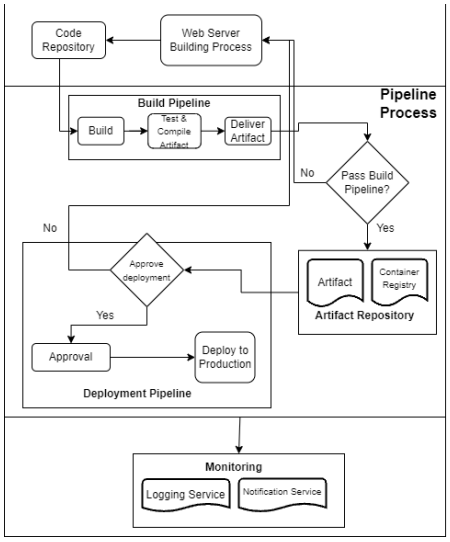
menentukan metode mana yang menghasilkan kinerja yang lebih optimal.

HASIL DAN PEMBAHASAN

1. *Plan*

   Dalam pengembangan web server, bahasa pemrograman yang digunakan adalah Python dengan framework Flask. Arsitektur web server dapat dilihat pada Gambar 6 di bawah ini. Web server akan menerima permintaan POST dari sistem AQG dan akan memberikan kumpulan pertanyaan dan jawaban yang dihasilkan oleh model AQG.

   Detail alur proses pembuatan pipeline menggunakan Docker dan Kubernetes dapat dilihat pada Gambar 1 di bawah. Web server akan di-*deploy* di Google Cloud Platform (GCP) menggunakan layanan Google Kubernetes Engine (GKE).



Gambar 1. Diagram alir CI/CD *Pipeline*

Model AQG dikembangkan menggunakan Python, OpenNMT, dan PyTorch dengan menggunakan *dataset* SQuAD versi 2.0. SQuAD berisi pertanyaan yang disusun oleh pekerja berbasis crowd dari artikel Wikipedia, dengan setiap pertanyaan memiliki jawaban dalam bentuk potongan teks relevan [18]. Model AQG dibuat di Google Colaboratory.

Model yang digunakan pada web server adalah BiLSTM *Uncased* karena memberikan hasil evaluasi terbaik dibandingkan dengan model BiGRU dan Transformer [2]. Hasil evaluasi BiLSTM *uncased* termasuk BLEU-1: 38.10, BLEU-2: 20.69, BLEU-3: 10.58, BLEU-4: 5.78, ROUGE-L: 42.98, dan METEOR: 18.31, yang merupakan hasil evaluasi tertinggi dibandingkan dengan konfigurasi lain.

2. *Code*

   Proses pengembangan web server untuk sistem AQG dilakukan di lingkungan pengembangan Visual Studio Code. Bahasa pemrograman Python digunakan sebagai alat pengembangan utama untuk web server dengan memanfaatkan *framework* Flask. Selain itu, langkah-langkah pengembangan kode akan disimpan dan dikelola melalui GitHub sebagai repositori kode dan sistem kontrol versi.

3. *Build*

   Pembangunan web server menggunakan Docker memfasilitasi proses pengemasan dan pengiriman aplikasi dalam kontainer. Docker beroperasi melalui perintah yang ditetapkan dalam Dockerfile dan docker-compose.yml. Dockerfile berisi instruksi yang diperlukan untuk membangun dan mengonfigurasi lingkungan kerja web server secara tepat, sementara docker-compose.yml digunakan untuk mengatur dan mengelola layanan web server dalam lingkungan kontainer yang terisolasi.

4. *Test*

   Unit test pada web server dijalankan menggunakan dependensi unittest, yang merupakan salah satu modul atau dependensi dari *framework* Flask. Proses *unit test* dilakukan untuk memastikan bahwa respons yang dihasilkan sesuai dengan yang diharapkan. Dengan implementasi *unit test*, web server dapat diuji secara sistematis setiap fungsi dan fiturnya, sehingga memastikan bahwa web server dapat beroperasi dengan baik dan sesuai dengan spesifikasi yang telah ditetapkan.

5. *Release*

   Setelah melewati tahap pengujian, web server dapat dirilis ke lingkungan produksi di Google Cloud Platform (GCP) dengan memanfaatkan layanan Google Kubernetes Engine (GKE).

6. *Deploy*

   Proses *deploy* dilakukan secara otomatis dengan bantuan GitHub Actions. Selain proses *deploy*, GitHub Actions juga membantu otomatisasi proses *build, test*, dan *release*. Proses *deploy* dilakukan melalui pembuatan *workflow* pada repositori GitHub. Dalam proses *deploy*, dilakukan *build* dan *test* untuk memastikan kualitas web server.

   Setelah web server berhasil di-*deploy* pada Google Kubernetes Engine (GKE), kinerja web server dapat dipantau untuk memastikan kelancaran operasinya. Evaluasi kinerja web server dilakukan secara terus-menerus untuk memastikan bahwa sistem berjalan dengan baik di lingkungan Google Kubernetes Engine.

7. *Operate*

   Proses *operate* dimulai dengan pembuatan Kubernetes Cluster di Google Kubernetes Engine, yang berfungsi sebagai lingkungan produksi untuk web server AQG. Setelah berhasil dibuat, langkah selanjutnya adalah menentukan *workloads* yang akan dijalankan di dalam *cluster*. *Workloads* ini akan dikelola oleh Kubernetes Cluster sebagai bagian dari infrastruktur yang mendukung operasi web server AQG. Selanjutnya, *workloads* tersebut akan dikonfigurasi untuk menjalankan web server AQG secara optimal di dalam *cluster*.

8. Monitor

   Web server yang telah diimplementasikan dapat dipantau secara terus-menerus untuk memastikan kinerjanya optimal. Proses pemantauan mencakup aspek performa, log aplikasi, dan pelacakan kesalahan yang mungkin terjadi pada aplikasi. Dengan adanya pemantauan yang berkelanjutan, tim pengelola dapat dengan cepat menanggapi masalah yang muncul dan memastikan bahwa web server beroperasi secara efisien dan dapat diandalkan.

EVALUASI CI/CD *PIPELINE*

Analisis waktu *deployment* menjadi kunci dalam mengevaluasi performa kedua metode. Dalam pengujian ini, CI/CD *pipeline* menunjukkan keunggulan dengan waktu *deploy* hanya 3 menit 40 detik, jauh lebih cepat dari metode tradisional yang memakan waktu 9 menit 1 detik seperti pada tabel 1.

Tabel 1. Total waktu *deployment*

Metode	Waktu *Deploy*.
CI/CD *Pipeline*	00:03:40
Metode Tradisional	00:09:01

Efisiensi dan efektivitas yang lebih baik dari CI/CD *pipeline* dapat meningkatkan respons sistem terhadap perubahan dengan penghematan waktu yang signifikan. Meskipun demikian, kompleksitas konfigurasi dan penyiapan lingkungan *virtual machine* mungkin menjadi faktor penyebab waktu *deploy* yang lebih lama pada metode *tradisional*.

Selain itu, dalam *load testing* dengan 5.000 dan 10.000 pengguna virtual menunjukkan hasil seperti tabel 2.

Tabel 2. Hasil Uji Beban

Metode	Pengguna Virtual	Rerata Waktu Respons	Sampel (Jumlah *Request*)	*Error rate*	*Throughput/s*
CI/CD *Pipeline*	5.000	10.03 s	45.184	6.28%	401.45/s
	10.000	7.83 s	99.412	5.61%	902.06/s
Metode Tradisional	5.000	5.92 s	48.311	4.02%	669.32/s
	10.000	12.34 s	94.583	46.24%	703.22/s

CI/CD *pipeline* menunjukkan peningkatan *throughput* sistem yang signifikan peningkatan *throughput* menandakan peningkatan kinerja sistem [19], penggunaan CI/CD *Pipeline* juga menurunkan rata-rata waktu respons sistem. Namun, metode tradisional menunjukkan kecenderungan sebaliknya, dengan peningkatan yang signifikan dalam rata-rata waktu respons sistem. Evaluasi performa sistem tidak hanya memperhitungkan aspek teknis seperti *throughput*, *error rate*, dan respons *time*, tetapi juga mempertimbangkan faktor waktu *deploy* dalam memilih metode *deployment* yang paling sesuai dengan kebutuhan lingkungan pengembangan dan produksi.

SIMPULAN

Berdasarkan hasil penelitian dan analisis, dapat disimpulkan bahwa implementasi penggunaan Docker dan Kubernetes pipeline melalui pendekatan MLOps telah berhasil meningkatkan efektivitas proses deployment. Penelitian dilaksanakan dengan menggunakan metode yang terstruktur, meliputi tahapan *Plan, Code & Build, Test, Release, Deploy & Operate*, dan Monitor. Terdapat peningkatan signifikan dalam efisiensi proses *deployment*, terbukti dengan penurunan waktu *deploy* sebesar 245% saat menggunakan metode DevOps, mengurangi waktu dari 9 menit 1 detik (pada metode tradisional) menjadi hanya 3 menit 40 detik.

Evaluasi kinerja antara metode penggunaan DevOps dan metode tradisional juga telah dilakukan. Proses evaluasi melibatkan penghitungan waktu deploy dan pelaksanaan *load testing* untuk membandingkan *error rate*, *throughput*, dan *response time*. Hasilnya menunjukkan bahwa penggunaan metode DevOps memberikan hasil yang lebih stabil dalam hal *error rate, throughput*, dan *response time* selama pengujian beban dengan 5.000 dan 10.000 pengguna virtual. Sebaliknya, metode tradisional menunjukkan peningkatan *error rate* dan *response time* aplikasi, terutama saat pengguna virtual ditingkatkan dari 5.000 menjadi 10.000.

SARAN

Berdasarkan simpulan dan keterbatasan produk dari penelitian, terdapat beberapa saran bagi penelitian mendatang sebagai berikut:

1. Disarankan untuk melakukan pengujian lebih lanjut yang melibatkan pembandingan antara metode DevOps dan metode tradisional melalui skenario *load testing* yang lebih kompleks dan realistis. Hal ini akan membantu dalam memperoleh pemahaman yang lebih mendalam tentang kinerja keduanya dalam kondisi yang lebih beragam.
2. Penting untuk mengidentifikasi dan menganalisis faktor-faktor yang memengaruhi performa kedua metode, termasuk konfigurasi infrastruktur, manajemen sumber daya, dan penyetelan parameter. Analisis lebih lanjut tentang faktor-faktor ini dapat memberikan wawasan tambahan untuk meningkatkan efektivitas dan efisiensi proses *deployment*.
3. Perlu dipertimbangkan juga aspek keamanan dan reliabilitas dalam kedua metode, termasuk keamanan kontainer. Penelitian lebih lanjut dalam hal ini dapat membantu mengidentifikasi potensi risiko dan mengembangkan strategi perlindungan yang lebih efektif bagi lingkungan *deployment*.

DAFTAR RUJUKAN

[1] V. A. Flores, L. Jasa, dan R. S. Hartati, "Pembangkit Pertanyaan Otomatis pada Materi Pelajaran Ilmu Pengetahuan Alam Berbahasa Indonesia di Tingkat Sekolah Dasar Berdasarkan Revisi Taksonomi Bloom," *Majalah Ilmiah Teknologi Elektro; Vol 20 No 2 (2021): (Juli-Desember) Majalah Ilmiah Teknologi Elektro*, 2021, doi: 10.24843/MITE.2021.v20i02.P19.

[2] Y. Indrihapsari *dkk.*, "A Comparison of OpenNMT Sequence Model for Indonesian Automatic Question Generation," *Elinvo (Electronics, Informatics, and Vocational Education)*, vol. 8, no. 1, hlm. 55–63, 2023.

[3] G. Kurdi, J. Leo, B. Parsia, U. Sattler, dan S. Al-Emari, "A systematic review of automatic question generation for educational purposes," *International Journal of Artificial Intelligence in Education*, vol. 30, hlm. 121–204, 2020.

[4] R. M. Elshiny dan A. Hamdy, "Automatic Question Generation Using Natural Language Processing and Transformers," dalam *2023 International Conference on Computer and Applications (ICCA)*, Nov 2023, hlm. 1–6. doi: 10.1109/ICCA59364.2023.10401848.

[5] O. Sharir, B. Peleg, dan Y. Shoham, "The Cost of Training NLP Models: A Concise Overview." 2020.

[6] E. Freeman, *DevOps For Dummies*. Wiley, 2019.

[7] D. Vela, A. Sharp, R. Zhang, T. Nguyen, A. Hoang, dan O. S. Pianykh, "Temporal quality degradation in AI models," *Scientific Reports*, vol. 12, no. 1, hlm. 11654, Jul 2022, doi: 10.1038/s41598-022-15245-z.

[8] L. Bass, I. Weber, dan L. Zhu, *DevOps: A software architect's perspective*. Addison-Wesley Professional, 2015.

[9] C. Ebert dan L. Hochstein, "DevOps in Practice," *IEEE Software*, vol. 40, no. 1, hlm. 29–36, Feb 2023, doi: 10.1109/MS.2022.3213285.

[10] C. Ebert, G. Gallardo, J. Hernantes, dan N. Serrano, "DevOps," *IEEE Software*, vol. 33, no. 3, hlm. 94–100, Jun 2016, doi: 10.1109/MS.2016.68.

[11] Dynantrace, "Deep cloud observability and advanced AIOps are key to scalling DevOps Practice 2021 Global DevOps Report," Global, 2021.

[12] T. Tohirin, S. F. Utami, S. R. Widianto, dan W. A. Mauludyansyah, "Implementasi DevOps Pada Pengembangan Aplikasi e-Skrining Covid-19.," *MULTINECTICS*, vol. 6, no. 1, hlm. 15–20, 2020, doi: https://doi.org/10.32722/multinetics.v6i1.2764.

[13] Y. Liu, Z. Ling, B. Huo, B. Wang, T. Chen, dan E. Mouine, "Building A Platform for Machine Learning Operations from Open Source Frameworks," *IFAC-PapersOnLine*, vol. 53, no. 5, hlm. 704–709, Jan 2020, doi: 10.1016/j.ifacol.2021.04.161.

[14] I. E. Fattoh, A. E. Aboutabl, dan M. H. Haggag, "Semantic question generation using artificial immunity," *International Journal of Modern Education and Computer Science*, vol. 7, no. 1, hlm. 1, 2015.

[15] S. A. I. B. S. Arachchi dan I. Perera, "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management,"

2018 Moratuwa Engineering Research Conference (MERCon), hlm. 156–161, Jul 2018, doi: 10.1109/MERCon.2018.8421965.

[16] M. R. Mufid, A. Basofi, M. U. H. Al Rasyid, I. F. Rochimansyah, dan A. rokhim, "Design an MVC Model using Python for Flask Framework Development," dalam *2019 International Electronics Symposium (IES)*, Sep 2019, hlm. 214–219. doi: 10.1109/ELECSYM.2019.8901656.

[17] I. Docker, "Docker," *Inea].[Junio de 2017]. Disponible en: https://www. docker. com/what-docker*, 2020.

[18] P. Rajpurkar, R. Jia, dan P. Liang, "Know What You Don't Know: Unanswerable Questions for SQuAD," *CoRR*, vol. abs/1806.03822, 2018, [Daring]. Tersedia pada: http://arxiv.org/abs/1806.03822

[19] D. I. Permatasari *dkk.*, "PENGUKURAN THROUGHPUT LOAD TESTING MENGGUNAKAN TEST CASE SAMPLING GORILLA TESTING," 2019. [Daring]. Tersedia pada: https://api.semanticscholar.org/CorpusID:214487469