BAB II

TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

Penelitian ini menggunakan sumber pustaka yang berhubungan dengan kasus yang diteliti, antara lain :

Tabel 2.1 Tinjauan Pustaka

No	Penulis	Kasus	Teknologi	Hasil
1	Parama et al., 2022	Implementasi Continuous Integration dan Continuous Delivery Pada Aplikasi myITS Single Sign On	Jenkins, Docker, Kubernetes, SonarQube	Waktu rata-rata berjalannya pipeline pada CI/CD Jenkins aplikasi MyITS Single Sign On adalah 355.4 detik atau sekitar 5,93 menit
2	Adrian et al., 2022	Optimasi Kecepatan Continuous Integration / Continuous Delivery pada Otomatisasi Jaringan dengan menggunakan algoritma Grey Wolf Optimizer.	Jenkins, algoritma Grey Wolf Optimizer	Implementasi algoritma GWO pada CI/CD dapat meningkatkan kecepatan deployment aplikasi sebesar 41,2%, dikarenakan algoritma ini membantu dalam mempercepat proses penemuan nilai tujuan
3	Alpery and Ridha, 2021	Implementasi CI/CD dalam Pengembangan	Docker, Jenkins	Rata-rata waktu yang dibutuhkan oleh pipeline untuk menyelesaikan proses

		Aplikasi Web Menggunakan Docker dan Jenkins.		deployment adalah 1 menit 58 detik, dengan tingkat keberhasilannya adalah sebesar 90%
4	Jaeni et al., 2022	Implementasi Continuous Integration/Continuous Delivery (CI/CD) pada Performance Testing DevOps	Jenkins ,Gitlab, Docker, ArgoCD, SonarQube, Kubernetes	Proses produksi perangkat lunak dengan menggunakan CI/CD dapat mempersingkat proses, meningkatkan kinerja, dan juga dapat memberikan hasil yang lebih teliti terkait penemuan bug-bug.
5	Wahyu and Noviantama , 2021	Implementasi Continuous Integration dan Continuous Deployment pada Aplikasi Learning Management System di PT. MILLENNIA SOLUSI INFORMATIKA	Gitlab, Enzyme, Docker	Implementasi konsep CI/CD memudahkan tim pengembang dan operasional untuk bekerja secara praktis. Otomatisasi pada CI/CD membuat kesalahan yang biasa terjadi karena manusia dapat dihindari karena seluruh proses dilakukan secara otomatis oleh mesin.

Persamaan penelitian ini dengan penelitian-penelitian di atas 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.

2.2 Dasar Teori

2.2.1 *Jenkins*

Jenkins adalah salah satu *tool open source* yang menyediakan layanan *CI* untuk pengembangan perangkat lunak. *Jenkins* merupakan tool yang sederhana, *user-friendly*, dan juga mudah untuk dikembangkan. *Jenkins* mendukung *tool SCM (Source Code Management)* seperti *StarTeam, Subversion, CVS, Git, AccuRev,* dan lain lain. Konsep *plugin* pada Jenkins membuatnya lebih atraktif, mudah untuk dipelajari dan juga mudah untuk digunakan. Terdapat berbagai macam kategori *plugin Jenkins* yang tersedia, diantaranya adalah *Source Code Management, Slave launchers and controllers, Build triggers, Build tools, Build notifies, Build reports, post-build actions, External site/tool integrations, UI Plugins, Authentication dan user management, Android development,* dan lain lain (Soni, 2015).

2.2.2 *Continuous Integration, Delivery*, dan *Deployment (CI/CD)*

Continuous Integration, *Delivery*, dan *Deployment* adalah praktik pengembangan yang relatif baru dan semakin populer dalam beberapa tahun terakhir. *Continuous Integration* berarti memeriksa perangkat lunak segera setelah ada perubahan dalam kode sumbernya, sehingga tujuannya adalah memastikan bahwa perangkat lunak berfungsi dengan baik setelah kode baru ditambahkan.

Continuous Delivery datang setelah *Continuous Integration* dan membuat proses penerapan perangkat lunak menjadi lebih mudah dan cepat, hingga perangkat lunak siap untuk digunakan dengan hanya sekali klik. *Continuous Deployment* kemudian datang setelah *Continuous Delivery* yang tujuannya adalah mengotomatisasi seluruh proses untuk mendeploy perangkat lunak kepada pelanggan atau menginstalnya di *server* yang digunakan (Rossel, n.d.).

2.2.3 *Container*

Container adalah sebuah format pengemasan dan distribusi standar yang umum digunakan, yang memungkinkan kapasitas pengangkutan yang jauh lebih besar, biaya lebih rendah, dan kemudahan dalam penggunaannya. Format kontainer ini berisi segala sesuatu yang diperlukan oleh aplikasi untuk berjalan, yang telah diintegrasikan ke dalam sebuah *file image* yang dapat dieksekusi oleh *container runtime* (Domingus and Arundel, n.d.).

2.2.4 *Docker*

Docker merupakan sebuah platform terbuka yang digunakan untuk mengembangkan, mengirimkan, serta menjalankan perangkat lunak. *Docker* dapat memisahkan perangkat lunak dari infrastruktur sehingga proses pengiriman perangkat lunak menjadi lebih cepat. Dengan menggunakan *Docker*, membuat proses mengelola infrastruktur sama seperti saat mengelola perangkat lunak. Dengan memanfaatkan kelebihan dari metode *Docker* untuk mengirimkan, menguji, serta merilis kode secara cepat, maka secara signifikan dapat

mengurangi penundaan antara menulis kode dengan menjalankannya di produksi. *Docker* memiliki kemampuan untuk membungkus dan menjalankan sebuah perangkat lunak dalam sebuah lingkungan terisolasi yang disebut dengan *Container*. Isolasi dan keamanan pada *Docker* memungkinkan untuk menjalankan banyak *Container* secara simultan dalam sebuah *host* yang ditentukan ("*Docker overview*," 2023).

2.2.5 *Amazon Web Service*

Amazon Web Services (AWS) adalah *platform* yang menyediakan berbagai layanan untuk untuk masalah komputasi, penyimpanan, dan jaringan dengan berbagai tingkat abstraksi yang berbeda. Kemampuan untuk meng-*hosting* situs web, menjalankan aplikasi *enterprise*, dan mengelola data dalam jumlah yang besar menjadi mungkin berkat adanya layanan-layanan *AWS* ini. *"Web service"* berarti layanan-layanan ini bisa dikendalikan melalui antarmuka web yang bisa digunakan oleh mesin atau manusia melalui tampilan grafis. Layanan-layanan yang populer adalah *EC2* yang menyediakan *server virtual*, dan *S3* yang menyediakan ruang penyimpanan. Layanan-layanan di *AWS* bekerja dengan baik bersama-sama, layanan *AWS* bisa digunakan untuk menduplikasi infrastruktur yang sudah ada atau merancang infrastruktur baru sesuai kebutuhan. Dan yang terbaik, layanan-layanan *AWS* hanya dibayar sesuai dengan penggunaan layanan yang digunakan (Wittig and Wittig, 2016).