PENGEMBANGAN *PROTOTYPE* ARSITEKTUR *CI/CD* PADA SISTEM INFORMASI MANAJEMEN TERINTEGRASI UNIVERSITAS PERJUANGAN MENGGUNAKAN JENKINS

Nizar Fadilah, Rudi Hartono, Dede Syahrul Anwar
Teknik Informatika, Universitas Perjuangan Tasikmalaya
J. Peta No. 177, Kahuripan, Kec.Tawang, Kab.Tasikmalaya, Jawa Barat 46115
nizarfadilah74@gmail.com

ABSTRAK

Universitas Perjuangan saat ini menghadapi permasalahan pada pengelolaan Sistem Informasi Manajemen Terintegrasi (SIMANTAP) yang masih menerapkan metode *deployment* secara manual. Pendekatan ini menyebabkan proses pengembangan perangkat lunak menjadi tidak efisien, rentan terhadap kesalahan, dan memperlambat pengiriman pembaruan aplikasi. Untuk mengatasi permasalahan tersebut, penelitian ini bertujuan mengembangkan prototipe arsitektur *Continuous Integration/Continuous Deployment* (*CI/CD*) pada SIMANTAP dengan menggunakan metode *Agile Kanban*. Metodologi yang diterapkan meliputi observasi, wawancara, studi pustaka, serta implementasi teknologi *Jenkins*, *Docker*, dan *SonarQube* yang terintegrasi dalam *pipeline CI/CD*. Evaluasi dilakukan melalui pengujian performa dan fungsionalitas, yang menunjukkan bahwa penerapan *CI/CD* mampu mengurangi waktu *deployment* hingga 10% dibandingkan metode manual. Selain itu, analisis kode menggunakan *SonarQube* berhasil mendeteksi bug dan kerentanan keamanan lebih awal sebelum aplikasi diterapkan ke lingkungan produksi. Hasil penelitian ini membuktikan bahwa implementasi *CI/CD* pada SIMANTAP meningkatkan efisiensi pengembangan perangkat lunak, mengurangi kesalahan *deployment*, serta mendukung pengiriman pembaruan aplikasi yang lebih cepat dan berkualitas.

Kata kunci : *CI/CD*, *Jenkins*, *SonarQube*, *Agile Kanban*, *Deployment* Otomatis

1. PENDAHULUAN

Dalam pengembangan perangkat lunak, praktik *Continuous Integration/Continuous Deployment* (*CI/CD*) menjadi kunci untuk mengotomasisasi integrasi dan pengiriman aplikasi secara cepat dan andal [1]. *CI/CD*, yang merupakan bagian dari pendekatan *DevOps* (*Development* dan *Operations*), bertujuan menghilangkan hambatan antara pengembangan dan operasional agar proses rilis perangkat lunak dapat dilakukan secara konsisten dan terkontrol. *Continuous Integration* (*CI*) memungkinkan setiap perubahan kode yang dibuat oleh tim pengembang untuk diuji dan diintegrasikan secara otomatis, sedangkan *Continuous Deployment* (*CD*) mengotomatisasi proses pengiriman perubahan kode tersebut ke lingkungan produksi [2].

Namun, di Universitas Perjuangan, pengelolaan Sistem Informasi Manajemen Terintegrasi Universitas Perjuangan (SIMANTAP) yang berfungsi mendukung aktivitas akademik mahasiswa masih bergantung pada proses *deployment* manual. Berdasarkan wawancara dengan tim pengembang SIMANTAP, diketahui bahwa setiap perubahan kode baru harus di*deploy* secara manual ke server produksi setelah dilakukan *push*/*commit* ke *repository*. Proses ini tidak hanya memakan waktu, tetapi juga meningkatkan risiko kesalahan *deployment* dan mengabaikan potensi kerentanan keamanan, karena pemeriksaan kode dilakukan tanpa sistem pengujian otomatis. Kondisi ini menghambat efisiensi kerja tim pengembang, memperlambat siklus permbaruan aplikasi, dan meningkatkan kemungkinan terjadinya gangguan pada layanan yang digunakan oleh mahasiswa.

Permasalahan tersebut menunjukkan urgensi penerapan *pipeline CI/CD* pada SIMANTAP untuk mengotomatisasi proses *build*, *testing*, dan *deployment*. Dengan pendekatan ini, potensi kesalahan dapat diminimalkan, kualitas kode dapat dipantau lebih ketat melalui analisis otomtais, dan waktu pengiriman pembaruan dapat dipercepat.

Untuk mendukung pengembangan *pipeline CI/CD* ini, digunakan metode *Agile Kanban*, yang mengutamakan visualisasi alur kerja, pengelolaan prioritas tugas, serta pembatasan jumlah pekerjaan dalam proses [3]. Pendekatan *Agile Kanban* memungkinkan tim pengembang SIMANTAP untuk lebih responsif terhadap perubahan kebutuhan pengembangan, mempercepat *feedback*, dan menjaga fokus pada pengiriman nilai yang berkelanjutan.

2. TINJAUAN PUSTAKA

Bab ini berisi pembahasan dasar-dasar teori atau penjelasan dari metode dan alat yang akan digunakan

2.1. *Jenkins*

Jenkins adalah sebuah alat *open source* yang banyak digunakan *developer* untuk melakukan prosses *Continuous Integration/Continuous Deployment* (*CI/CD*) pada pengembangan perangkat lunak [4]. *Jenkins* akan Smelakukan proses *deployment* setelah *developer* melakukan perubahan kode di *repository*. Pada proses ini, *jenkins* akan mengambil kode terbaru dan melakukan proses *deployment* sesuai dengan instruksi yang diberikan [5].

2.2. Pengembangan *Prototype*

Menurut [6], pengembangan *prototype* merupakan suatu metode pengembangan perangkat lunak yang menghasilkan model fisik kerja sistem sebagai versi awal dari sistem. Model ini berfungsi sebagai perantara antara pengembang dan pengguna sehingga mereka dapat berinteraksi dalam proses pengembangan sistem informasi.

2.3. *CI/CD Pipeline*

CI/CD Pipeline adalah serangkaian proses yang mendukung integrasi perubahan kode dan *deployment* ke lingkungan produksi [7]. *CI/CD* adalah singkatan dari *Continuous Integration* dan *Continuous Deployment*. Kedua konsep ini bekerja untuk mempercepat proses pengembangan perangkat lunak, memastikan bahwa perubahan kode yang telah dibuat oleh pengembang dapat diintegrasikan, diuji, dan di *deploy* ke dalam lingkungan produksi secara otomatis.

2.4. *Agile Development*

Agile Development adalah pendekatan dalam pengembangan perangkat lunak yang menitikberatkan pada fleksibilitas serta kerjasama antara tim pengembang dan pengguna [8]. Metode ini dirancang untuk merespon perubahan kebutuhan secara cepat dan memungkinkan peningkatan berkelanjutan pada perangkat lunak yang sedang dikembangkan [9].

2.5. *Kanban*

Kata *Kanban* berasal dari dua kata dalam bahasa Jepang, yaitu *Kan* yang berarti visual dan *Ban* yang berarti kartu atau papan. Secara umum, *Kanban* dapat diartikan sebagai papan tanda [10]. Didalam pengembangan perangkat lunak, *kanban* bertujuan untuk meningkatkan efisiensi dan produktivitas dengan memperlihatkan proses kerja dan memantau status setiap tugas [11]. Dengan menggunakan *kanban*, pekerjaan dipecah menjadi tugas-tugas kecil yang kemudian dipetakan pada papan visual, biasanya dibagi ke dalam kolom-kolom seperti "*To Do*"*, "In Progress"*, "*Review*", "*Done*", "*Done*". Metode ini memungkinkan tim untuk fokus pada penyelesaian tugas sebelum memulai yang baru, sehingga meminimalkan multitasking dan memastikan setiap tugas selesai tepat waktu [12].

2.6. *Jira*

Jira adalah perangkat lunak yang dikembangkan oleh perusahaan *Atlassian*. Perangkat ini banyak digunakan oleh tim pengembang untuk mengelola proyek, memantau perkembangan sebuah proyek, dan berkolaborasi secara *real-time* [13].

2.7. *Git*

Git adalah salah satu alat *Version Control System* (*VCS*) yang dirancang untuk mempermudah pengembang dalam mengelola perubahan terhadap kumpulan file dalam sebuah proyek [14]. *VCS* bertugas untuk mencatat setiap perubahan file proyek yang sudah dikerjakan dan dapat melihat, membandingkan, dan mengembalikan versi file ke perubahan yang berbeda.

2.8. *Github*

Github merupakan sebuah layanan berbasis *cloud* yang menyediakan fasilitas untuk kolaborasi, dokumentasi, serta versi kontrol dalam pengelolaan kode dan data penelitian. Dalam penelitian ilmiah, terutama di bidang teknologi dan evolusi, *Github* memainkan peran penting dalam mendukung transparansi, replikasi, serta efisiensi dalam pengolahan analisis data [15].

2.9. SonarQube

SonarQube adalah salah satu perangkat lunak *open source* yang digunakan untuk melakukan pemeriksaan kode. Lebih dari itu, *SonarQube* juga digunakan untuk mendeteksi *detec defects, vulnerabillities*, dan *code smells* dalam 30 bahasa pemrograman [16]. *SonarQube* mempunyai beberapa versi yaitu, *Community Edition, Developer Edition, Enterprise,* dan *Data Center*.

2.10. *Amazon Web Services* (*AWS*)

Amazon Web Services (*AWS*) adalah platform penyedia layanan komputasi awan yang dikembangkan oleh *Amazon*. *AWS* menyediakan berbagai layanan berbasis *cloud* yang dapat digunakan untuk penyimpanan data berbasis awan, layanan *database,* analisis, *IoT*, *mobile computing*, dan *enterprise services* [17].

2.11. Docker

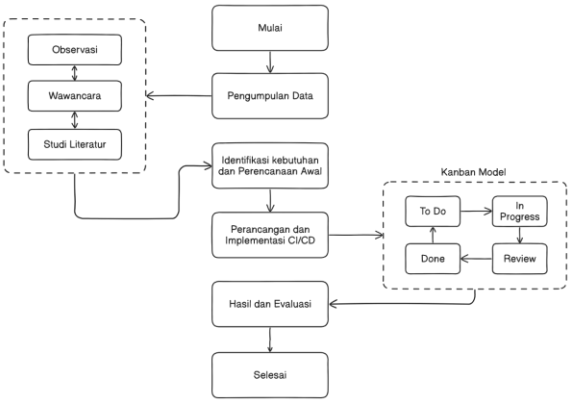
Docker adalah teknologi yang memungkinkan untuk mengenkapsulasi aplikasi dengan menggunakan *container* [18]. *Container* adalah wadah virtual yang dapat mengemas aplikasi dan semua *environment* sehingga aplikasi dapat berjalan di berbagai lingkungan mulai dari lokal hingga server *cloud*

3. METODE PENELITIAN

Bab ini menguraikan alat, dan tahapan penelitian dalam pengembangan *prototype* arsitektur *Continuous Integration/Continuous Deployment* (*CI/CD*) pada Sistem Informasi Manajemen Terintegrasi Universitas Perjuangan (SIMANTAP).

3.1. Tahapan Penelitian

Tahapan penelitian ini dibagi menjadi beberapa tahapan yang akan dilaksanakan dari awal hingga akhir yang akan digambarkan dalam flowchart untuk menjelaskan alur penelitian



Gambar 1. Tahapan Penelitian

Gambar 1 merupakan tahapan penelitian yang akan dilaksanakan dari awal hingga akhir. Gambar berikut mencakup tahap pengumpulan data, identifikasi kebutuhan dan perancanaan awal, perancangan dan implementasi *CI/CD*, hasil dan evaluasi

3.2. Pengumpulan Data

Pengumpulan data yang diperlukan untuk penelitian ini memiliki beberapa tahapan yaitu

a. Observasi
   Hasil dari observasi menunjukkan bahwa proses *deployment* masih dilakukan secara manual, mulai dari pengiriman kode masih secara manual.
b. Wawancara
   Hasil wawancara mengungkapkan beberapa kebutuhan utama tim pengembang diantaranya adalah, pengujian otomatis, *monitoring* dan *logging*, dan automasi *deployment*
c. Studi Literatur
   Studi literatur dilakukan untuk memahami teori dan metode terkait *CI/CD* termasuk penggunaan alat seperti *jenkins*, *sonarqube*, dan *docker*.

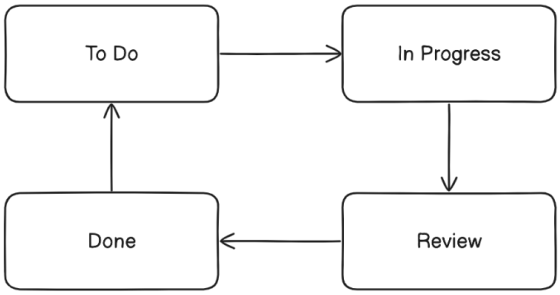
3.3. Identifikasi Kebutuhan dan Perencanaan Awal

Berdasarkan hasil observasi dan wawancara, beberapa kebutuhan utama yang harus dipenuhi dalam proses integrasi *deployment* sistem SIMANTAP adalah sebagai berikut:

a. Otomatisasi *Deployment*
   Diperlukan mekanisme otomatisasi dalam proses *deployment* untuk mengurangi kesalahan manusia dan meningkatkan efisiensi.
b. Pemeriksaan Kode Otomatis
   Implementasi alat yang dapat melakukan analisis kode secara otomatis untuk mengurangi potensi kerentanan keamanan
c. Monitoring dan *Logging*
   Penggunaan alat pemantauan *real-time* untuk mendeteksi kesalahan pada proses *deployment*

3.4. Perancangan dan Implementasi CI/CD

Berdasarkan perencanaan yang telah dibuat, tahap ini berfokus pada perancangan serta implementasi arsitektur *CI/CD*. Dalam perancangan dan implementasi *CI/CD* pada SIMANTAP, metode *Agile Kanban* diterapkan untuk mengelola alur kerja pengembangan secara efisien dan adaptif. *Agile Kanban* dipilih karena kemampuannya dalam mempermudah pengembangan *pipeline CI/CD* pada SIMANTAP. *Agile Kanban* memungkinkan tim untuk mengelola proses pengembangan secara terstruktur dan transparan melalui papan *Kanban* yang memberikan tampilan visual dari setiap tahapan pekerjaan. Papan *kanban* yang digunakan dalam pengembangan *pipeline CI/CD* ini terdiri dari beberapa kolom yang digambarkan pada gambar dibawah ini:



Gambar 2. Papan Kanban

Pada gambar 2 merupakan papan *kanban* yang akan digunakan dalam penelitian ini. Dalam kolom papan *kanban* mempunyai beberapa item yaitu

a. *To Do*
   Kolom ini berisi daftar tugas yang telah diidentifikasi dan akan dikerjakan, beserta beberapa atribut seperti label prioritas dan tenggat waktu pengerjaan
b. *In Progress*
   Setelah tugas diambil oleh anggota tim, kartu tugas akan dipindahkan ke kolom *In* Progress. Tugas di kolom ini sedang dikerjakan oleh tim dan dipantau secara aktif
c. *Review*
   Setelah selesai dikerjakan, tugas akan dipindahkan ke kolom *review.* Pada tahap ini, tugas yang telah diselesaikan akan ditinjau oleh anggota tim lain atau pengembangan senior untuk memastikan kualitas kerja sebelum dipindahkan ke tahap *deployment*
d. *Done*
   Tugas yang sudah melewati proses *review* dan dianggap memenuhi standar akan dipindahkan ke kolom *done*. Ini menandakan bahwa tugas tersebut telah selesai dan tidak memerlukan tindakan lebih lanjut

3.5. Hasil dan Evaluasi

Setelah implementasi selesai, evaluasi terhadap *prototype* arsitektur *CI/CD* dilakukan melalui dua metode pengujian, yaitu pengujian performa dan pengujian fungsionalitas

a. Pengujian Performa
   Pengujian ini bertujuan untuk mengukur kecepatan, efisiensi, dan stabilitas *pipeline CI/CD* yang dikembangkan. Beberapa metrik yang digunakan dalam pengujian ini meliputi waktu *build*, waktu *deployment*, serta penggunaan sumber daya sistem selama proses *CI/CD* berlangsung
b. Pengujian Fungsionalitas
   Pengujian ini dilakukan untuk memastikan bahwa *pipeline CI/CD* berfungsi sesuai dengan spesifikasi yang dirancang. Setiap tahapan dalam *pipeline*, mulai dari proses *build*, *testing*, hingga *deployment*, diuji untuk memastikan bahwa sistem berjalan dengan baik dan menghasilkan *output* yang sesuai dengan yang diharapkan

3.6. Alat dan Teknologi yang Digunakan

Implementasi *pipeline CI/CD* dalam penelitian ini membutuhkan beberapa alat dan teknologi berikut ini

a. Jenkins (v2.426.3)
   Jenkins merupakan alat utama yang digunakan untuk optimalisasi *pipeline CI/CD*. Pada penelitian ini, *jenkins* versi 2.426.3 digunakan untuk mengatur proses otomatisasi dalam *build*, *testing*, dan *deployment*
b. Docker (v24.0.7)
   Docker versi 24.0.7 digunakan untuk mengisolasi aplikasi dalam *container*, memungkinkan aplikasi berjalan secara konsisten di berbagai lingkungan
c. *SonarQube Community Edition* (v25.3)
   SonarQube Community Edition versi 25.3 digunakan sebagai alat analisis kode untuk memastikan kualitas perangkat lunak. Versi ini memiliki fitur utama seperti deteksi otomatis terhadap *bug*, *vulnerabillities*, dan *code smells*, serta dukungan untuk berbagai bahasa pemrograman
d. *Amazon Web Services* (*AWS*)
   Menyediakan layanan *cloud* untuk *deployment* aplikasi ke lingkungan produksi
e. *Github*
   Digunakan sebagai alat manajemen *repository* kode yang akan mengintegrasikan *pipeline CI/CD* dengan *jenkins*
f. *Slack* (*Desktop* v4.36.130)
   Slack digunakan alat komunikasi tim yang terintegrasi dalam *pipeline CI/CD*. Dalam penelitian ini, *Slack* versi 4.36.139 digunakan untuk mengirimkan notifikasi otomatis terkait status *pipeline*, seperti keberhasilan atau kegagalan pada proses *deployment*

4. HASIL DAN PEMBAHASAN

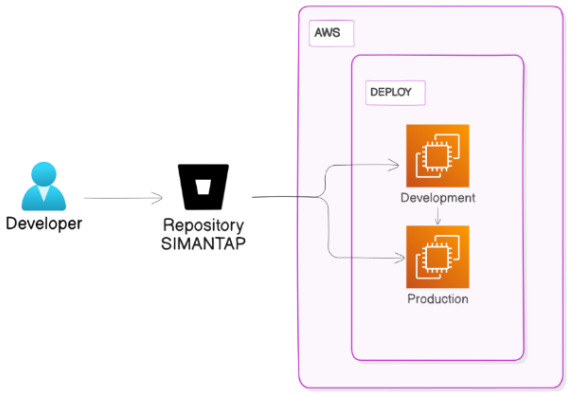
Bab ini menjelaskan implementasi dan hasil pengembangan *prototype* arsitektur *CI/CD* pada Sistem Infomrasi Manajemen Terintegrasi Universitas Perjuangan

4.1. Pengumpulan Data

Pengumpulan data dalam penelitian ini dilakukan dengan memanfaatkan metode *Agile Kanban* untuk memantau dan mengelola alur kerja secara visual dan sistematis. Papan *Kanban* digunakan sebagai alat bantu utama untuk melacak setiap tahapan tugas, mulai dari identifikasi kebutuhan hingga implementasi.

4.2. Proses Arsitektur yang Berjalan pada SIMANTAP

Arsitektur Sistem Informasi Manajemen Terintegrasi Universitas Perjuangan (SIMANTAP) yang berjalan saat ini memiliki beberapa kekurangan pada proses *deployment* aplikasi. Berikut adalah gambaran arsitektur *deployment* yang berjalan pada SIMANTAP



Gambar 3. Arsitektur *Deployment* SIMANTAP

Gambar 3 merupakan gambar arsitektur *deployment* SIMANTAP yang sedang berjalan sekarang, dimana proses *deployment* versi sekarang memiliki kelemahan, khususnya pada tahapan *deployment* aplikasi ke server. Pada arsitektur ini, *developer* melakukan pengunggahan aplikasi secara manual ke server, yang mengakibatkan potensi terjadinya kesalahan, kurang efisien, serta memerlukan waktu yang cukup lama untuk setiap perubahan aplikasi

4.3. Perancangan Arsitektur

Pada tahap ini, dilakukan perancangan arsitektur *CI/CD* Sistem Informasi Manajemen Terintegrasi Universitas Perjuangan (SIMANTAP) dengan tujuan meningkatkan efisiensi dan kecepatan proses pengembangan serta *deployment* aplikasi. Berikut adalah tahapan dalam perancangan arsitektur *CI/CD* yang akan diterapkan

4.4. Analisis Kebutuhan

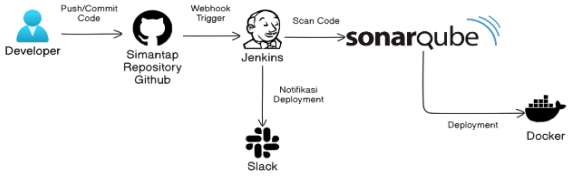
Analisis kebutuhan dilakukan untuk memahami tantangan dalam proses *deployment* yang ada serta menentukan alat dan teknologi yang akan digunakan. Beberapa kebutuhan utama yang diidentifikasi meliputi

a. Otomatisasi *build*, *testing*, dan *deployment*
b. Integrasi dengan *repository* kode
c. Pemantauan dan analisis kode secara otomatis

d. Penerapan kontainerisasi untuk memudahkan *deployment*
e. Pemberitahuan otomatis kepada tim pengembang

4.5. Pemilihan Teknologi

Pemilihan teknologi digambarkan dengan gambar berikut yang menunjukkan bagaimana setiap komponen bekerja dalam ekosistem *CI/CD*



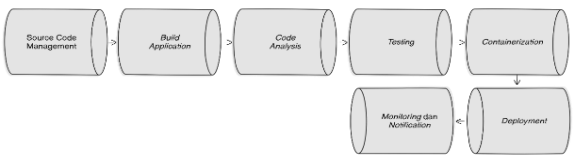
Gambar 4. Teknologi yang Digunakan

Berdasarkan gambar 4 teknologi yang digunakan dalam implementasi *CI/CD* pada SIMANTAP meliputi

a. Jenkins sebagai alat utama yang mengotomatisasi pipeline *CI/CD*. *Jenkins* mendeteksi perubahan kode dari *repository github* dan menjalankan tahapan *build*, *testing*, serta *deployment* secara otomatis
b. *Github* digunakan sebagai sistem manajemen kode sumber. Setiap perubahan kode yang dilakukan pengembang akan memicu *pipeline ci/cd* melalui *webhook*
c. *Sonarqube* berfungsi sebagai analisis kualitas kode untuk mendeteksi *bug*, code *smells*, dan *vulnerabillities* sebelum kode di *deploy*
d. *Docker* digunakan untuk mengemas aplikasi ke dalam kontainer guna memastikan dapat berjalan secara konsisten diberbagai lingkungan
e. *Slack* digunakan untuk memberikan notifikasi otomatis kepada tim pengembang mengenai status *deployment*, sehingga jika terjadi kegagalan, tim dapat segera melakukan perbaikan

4.6. Perancangan *Pipeline CI/CD*

Pipeline CI/CD dirancang untuk memastikan bahwa setiap perubahan kode yang dilakukan oleh pengembang akan melalui beberapa tahap sebelum akhirnya di*deploy* ke lingkungan produksi. Berikut adalah ilustrasi tahapan *pipeline ci/cd*



Gambar 5. Rancangan *Pipeline CI/CD*

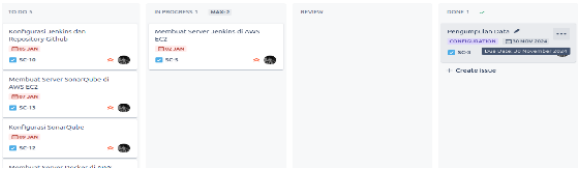
Gambar 5 merupakan rancangan *pipeline* yang dirancang untuk memastikan bahwa setiap kode yang dilakukan oleh pengembang akan melalui tahapan otomatis sebelum diterapkan ke lingkungan produksi. *Pipeline* ini mencakup tahapan seperti *source code management*, *build*, *code analysis*, *testing*, *conterization*, *deployment*, dan *notification*

4.7. Implementasi Arsitektur CI/CD

Dalam tahap ini akan mengimplementasikan arsitektur *CI/CD* yang sudah dirancang. Untuk mengimplementasikan rancangan *CI/CD* yang sudah dibuat, penulis menggunakan Papan *Kanban* sebagai alat bantu proses implementasi.

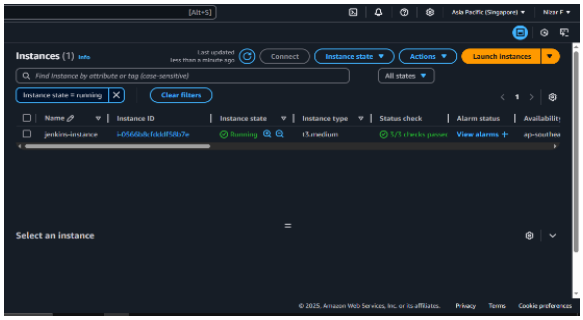
a. Membuat Server *Jenkins* di *AWS EC 2*

Dalam proses pembuatan server *Jenkins* di *AWS EC2*, tugas yang berada pada tabel kanban maka akan dipindahkan ke kolom in progress



Gambar 6. Tugas membuat server *jenkins* di Papan *Kanban*

Gambar 6 merupakan tugas membuat server *jenkins* di *aws ec2* dipindahkan ke kolom in progress. Berikut adalah hasil pembuatan server *jenkins* di *aws ec2*

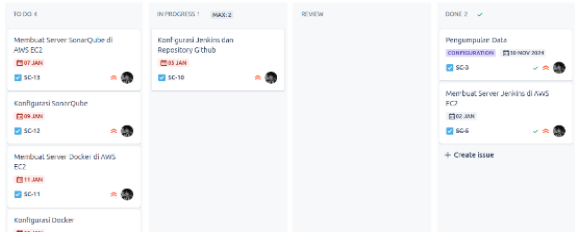


Gambar 7. Hasil Pembuatan Server *Jenkins* di *AWS EC2*

Gambar 7 merupakan hasil dari pembuatan server *jenkins* di *aws ec2*. Setelah berhasil melakukan pembuatan server, terlihat bahwa server dengan *jenkins-instance* telah berhasil dijalankan

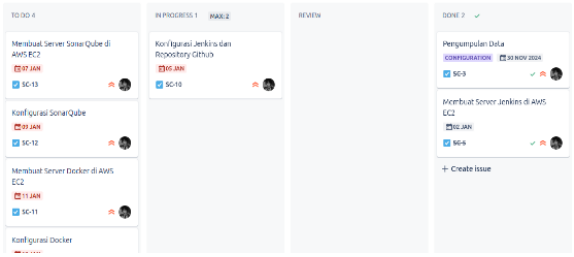
b. Konfigurasi *Jenkins* dan *Repository Github*

Dalam proses konfigurasi *jenkins* dan *repository github*, tugas yang berada pada tabel *kanban* maka akan dipindahkan ke kolom in progress



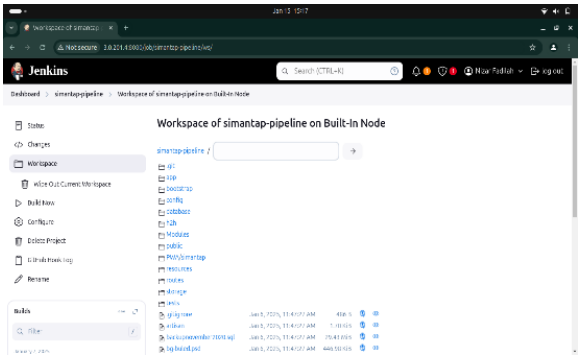
Gambar 8. Tugas konfigurasi *jenkins* dan *repository github*

Gambar 8 merupakan tugas konfigurasi *jenkins* dan *repository github* dipindahkan ke kolom *in progress*. Berikut adalah hasil dari konfigurasi *jenkins* dan *repository github*



Gambar 9. Tampilahan halaman *web jenkins*

Gambar 9 merupakan hasil dari konfigurasi *jenkins*. Pada gambar 9 terlihat tampilan awal *dashboard jenkins* setelah di proses instalasi dan konfigurasi selesai dilakukan. Proses konfigurasi mencakup instalasi *jenkins* pada server ,pemasangan *plugin* yang diperlukan, serta pembuatan akun admin untuk mengakses dan mengelola *jenkins*. Selanjutnya adalah melakukan konfigurasi *repository github* sehingga setiap perubahan kode yang dilakukan pada *repository* dapat terdeteksi dan diproses secara otomatis oleh *jenkins*. Berikut adalah hasil dari konfigurasi *repository github*

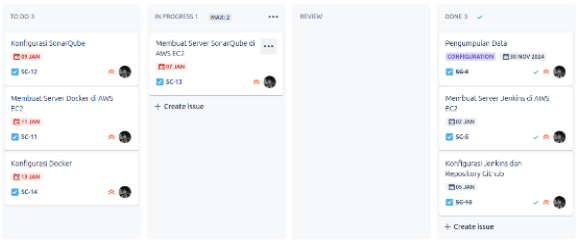


Gambar 10. *Repository github* berhasil diintegrasikan dengan *jenkins*

Gambar 10 merupakan *repository* yang sudah berhasil diintegrasikan dengan *jenkins*. Ketika *repository* kode berhasil diintegrasikan, *jenkins* dapat secara otomatis mendeteksi setiap perubahan kode yang dilakukan pada *repository*. Perubahan kode tersebut dapat dilihat melalui menu *changes*, yang menampilkan daftar *commit* terbaru, termasuk informasi tentang siap yang melakukan perubahan, waktu perubahan, dan deskripsi dari setiap *commit*

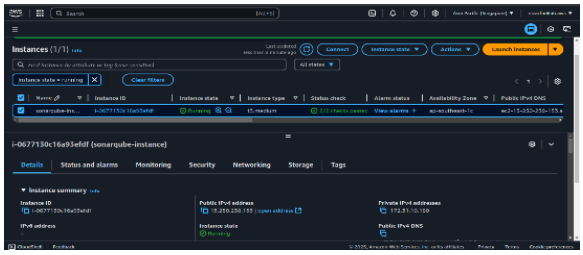
c. Membuat Server *SonarQube* di *AWS EC2*

Dalam proses pembuatan server *sonarqube* di *aws ec2*, tugas yang berada pada tabel *kanban* maka akan dipindakan ke kolom *in progress*



Gambar 11. Tugas membuat server *sonarqube* di *aws ec2*

Gambar 11 merupakan tugas membuat server *sonarqube* di *aws ec2* dipindahkan ke kolom *in progress*. Berikut adalah hasil pembuatan server *sonarqube* di *aws ec2*

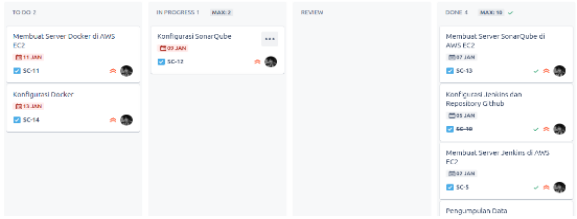


Gambar 12. Hasil pembuatan server *sonarqube* di *AWS EC2*

Gambar 12 merupakan hasil dari pembuatan server *sonarqube* di *aws ec2*. Setelah berhasil melakukan pembuatan server, terlihat bahwa server dengan *sonarqube-instance* telah berhasil dijalankan

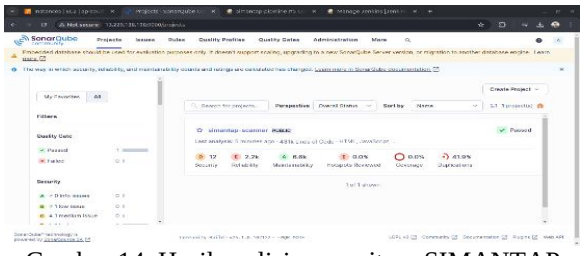
d. Konfigurasi *SonarQube*

Dalam proses konfigurasi *SonarQube*, tugas yang berada pada tabel kanban maka akan dipindahkan ke kolom *in progress*



Gambar 13. Tugas konfigurasi *SonarQube*

Gambar 13 merupakan tugas konfigurasi *SonarQube* dipindahkan ke kolom in progress. Berikut adalah hasil analisis *code* program SIMANTAP menggunakan *SonarQube*

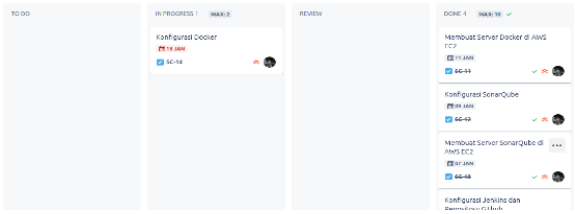


Gambar 14. Hasil analisis *repository* SIMANTAP

Gambar 14 merupakan hasil analisis yang dilakukan pada *repository* SIMANTAP. Analisis ini menunjukkan status kualitas kode berdasarkan metrik utama, seperti keamanan (*security*), keandalan (*reliability*), dan tingkat duplikasi kode (*duplication*). Hasil analisis menunjukkan terdapat 12 isu keamanan yang memerlukan perhatian, 2.2 ribu isu reliabilitas, tingkat duplikasi kode yang tinggi sebesar 41.9%, namun dengan kemampuan pemeliharaan yang baik di kategori A

e. Konfigurasi *Docker*

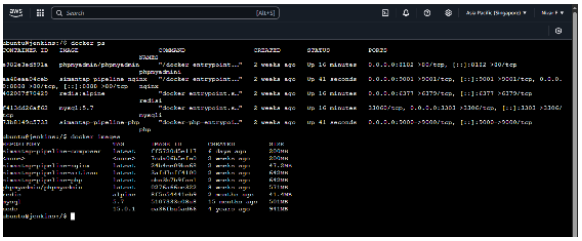
Dalam proses konfigurasi *docker*, tugas yang berada pada tabel kanban maka akan dipindahkan ke kolom *in progress*



Gambar 15. Tugas konfigurasi *Docker*

Gambar 15 merupakan tugas konfigurasi *Docker* dipindahkan ke kolom *in progress*. Berikut adalah hasil dari konfigurasi *docker*

Pada konfigurasi ini mencakup penginstalan *docker*, *docker-compose* dan konfigurasi *docker-compose*.

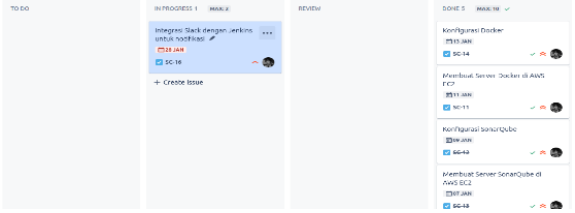


Gambar 16. Hasil konfigurasi *docker*

Pada gambar 16 merupakan hasil konfigurasi *docker*. Pada konfigurasi ini mencakup menginstalan *docker*, *docker-compose*, dan konfigurasi *docker-compose.*

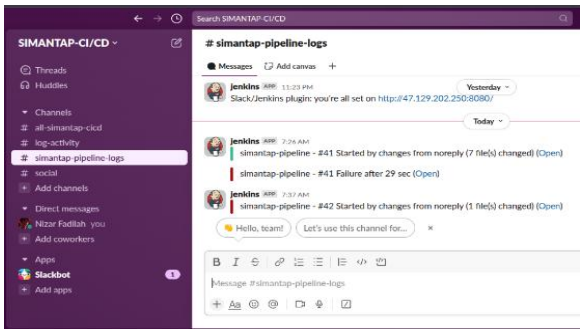
f. Integrasi *Slack*

Dalam proses integrasi *Slack* dengan *jenkins* untuk notifikasi, tugas yang berada pada tabel *kanban* akan dipindahkan ke kolom *in progress*



Gambar 17. Tugas integrasi *slack* dengan *jenkins*

Gambar 17 merupakan tugas integrasi *slack* dengan *jenkins* dipindahkan ke kolom *in progress*. Berikut adalah hasil dari integrasi *slack* dengan *jenkins*



Gambar 18. Hasil integrasi *slack* dengan *jenkins*

Gambar 18 merupakan hasil dari integrasi *slack* dengan *jenkins*. Dalam implementasi ini, *jenkins* di konfigurasi dengan *plugin slack notification* yang memungkinkan komunikasi langsung dengan *workspace slack* yang digunakan oleh tim pengembang. Setiap *pipeline* yang berjalan akan mengirimkan laporan status, termasuk informasi apakah *build* berhasil atau mengalami kegagalan

4.8. Pengujian dan Evaluasi

Evaluasi terhadap implementasi arsitektur *CI/CD* pada Sistem Informasi Manajemen Terintegrasi Universitas Perjuangan (SIMANTAP) dilakukan melalui dua metode pengujian, yaitu pengujian performa dan pengujian fungsionalitas

1. Pengujian Performa

a. Waktu *deployment*

Untuk mengukur waktu yang dibutuhkan *jenkins* melakukan *build* terhadap kode yang telah dikirimkan ke *repository github*, pengembang melakukan perubahan kode *repository* kode. Hasil pengukuran waktu *build* dapat dilihat pada tabel berikut ini

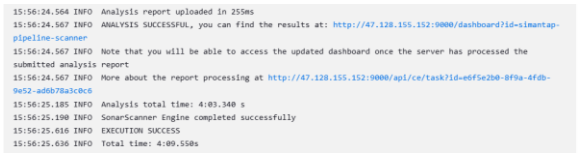
Tabel 1. Hasil waktu proses *build*

		Primary Task	***Including Substask***
In Queue	***Waiting***	9.9 *sec*	9.9 *sec*
	Blocked	0 *ms*	0 *ms*
	Buidable	18 *ms*	18 *ms*
	Total	9.9 *sec*	9.9 *sec*
Building		4 min 29 *sec*	4 min 29 *sec*

Tabel 1 merupakan hasil proses *build* yang dihasilkan ketika pengembang melakukan perubahan kode pada *repository*. Pada gambar tersebut, proses *build* memerlukan waktu 4 menit 29 detik, sedangkan jika dilakukan secara manual, proses yang sama membutuhkan waktu hingga 5 menit. Hal ini menunjukkan bahwa penerapan *pipeline CI/CD* menggunakan *jenkins* mampu mengurangi proses *build* sekitar 10% dibandingkan dengan metode manual

b. Waktu scan kode menggunakan *sonarqube*

Proses analisis kode menggunakan sonarqube dilakukan setelah tahap *build* untuk mendeteksi *bug*, *vulnerabillities*, dan *code smells*. Dari hasil pengujian, waktu yang dibutuhkan *sonarqube* untuk *scanning* terhadap *repository* SIMANTAP dapat dilihat pada gambar berikut ini



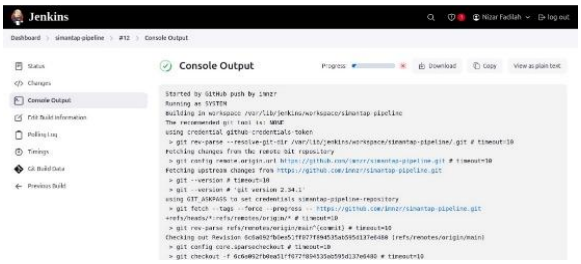
Gambar *19. Hasil waktu proses scan menggunakan sonarQube*

Gambar 19 menunjukkan hasil eksekusi analisis kode menggunakan *sonarqube*, dimana total waktu yang dibutuhkan untuk proses analisis adalah 4 menit 9 detik. Informasi ini menunjukkan bahwa proses analisis kode berjalan dengan sukses, dan hasilnya dapat diakses melalui *dashboard sonarqube* untuk evaluasi lebih lanjut

2. Pengujian Fungsionalitas

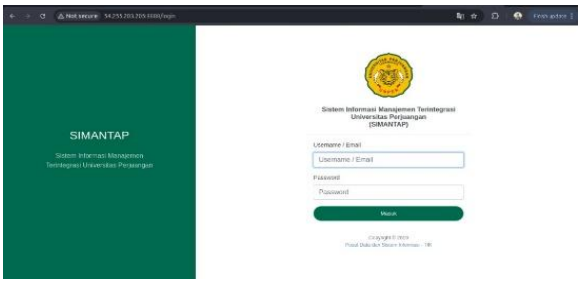
a. Pengujian *build*

Pengujian *build* dilakukan dengan mengubah kode program pada *repository* SIMANTAP di *github*. Setelah melakukan perubahan pada *repository*, *jenkins* akan terpicu seperti gambar yang ditunjukkan pada gambar 20



Gambar 20. *Log jenkins*

menunjukkan *jenkins* melakukan *pull code* sesuai dengan kode yang ter-*update*



Gambar 21. Antarmuka aplikasi

Setelah *deployment* selesai, dilakukan pengujian antarmuka untuk memastikan aplikasi berjalan dengan baik. Gambar 21 menunjukkan bahwa pengujian telah berhasil dilakukan dan aplikasi telah berhasil di *deploy*. Hal ini ditandai dengan tampilan antarmuka aplikasi yang berfungsi sebagai mestinya tanpa adanya kendala atau *error* dalam proses pengujian

b. Pengujian analisis kode dengan *sonarqube*

Gambar 22. Hasil analisis repository SIMANTAP

Gambar 22 merupakan hasil analisis yang dilakukan pada *repository* SIMANTAP. Analisis ini menunjukkan status kualitas kode berdasarkan metrik utama, seperti keamanan (*security*), keandalan (*reliability*), dan tingkat duplikasi kode (*duplication*). Hasil analisis menunjukkan terdapat 12 isu keamanan yang memerlukan perhatian, 2.2 ribu isu reliabilitas, tingkat duplikasi kode yang tinggi sebesar 41.9%, namun dengan kemampuan pemeliharaan yang baik di kategori A

c. Pengujian notifikasi dengan *slack*

Gambar 23. Notifikasi pipeline di saluran slack

Gambar 23 menunjukkan hasil integrasi antara *jenkins* dan *slack* dalam saluran *#simantap-cicd-logs*. Pada gambar tersebut, terlihat bahwa *jenkins* mengirimkan notifikasi otomatis setiap kali *pipeline* berjalan, termasuk informasi tentang perubahan kode, nomor *pipeline* (#13 dan #14) , serta status eksekusi. Dengan adanya notifikasi ini, tim pengembang dapat memantau proses *CI/CD* secara *real-time* dan segera mengambil tindakan jika terjadi kegagalan dalam *pipeline*

5. KESIMPULAN DAN SARAN

Berdasarkan hasil penelitian yang telah dilakukan terkait pengembangan dan evaluasi arsitektur Continuous Integration/Continuous Deployment (CI/CD) pada Sistem Informasi Manajemen Terintegrasi Universitas Perjuangan (SIMANTAP), dapat disimpulkan bahwa pengembangan prototipe arsitektur CI/CD berhasil diimplementasikan melalui integrasi teknologi Jenkins, Docker, SonarQube, dan AWS EC2, dengan

dukungan metodologi agile kanban yang berkontribusi dalam meningkatkan transparansi serta efisiensi pengelolaan proyek. Evaluasi implementasi menunjukkan bahwa penerapan CI/CD mampu meningkatkan efisiensi proses deployment aplikasi dengan pengurangan waktu build sebesar 10% dibandingkan metode manual, serta memungkinkan proses build, pengujian, analisis kualitas kode, dan deployment dilakukan secara otomatis. Selain itu, penerapan CI/CD memberikan manfaat signifikan dalam mendeteksi kesalahan kode pada tahap awal, serta mempermudah monitoring dan notifikasi bagi tim pengembang. Sehubungan dengan temuan tersebut, direkomendasikan agar dilakukan optimalisasi infrastruktur dengan penggunaan server berkapasitas lebih tinggi untuk mempercepat proses build dan deployment, peningkatan keamanan kode melalui penerapan standar pengkodean yang baik dan audit kode secara berkala berdasarkan hasil analisis SonarQube, serta evaluasi rutin terhadap performa pipeline CI/CD guna memastikan efektivitas dan mengidentifikasi peluang perbaikan secara berkelanjutan

DAFTAR PUSTAKA

[1] A. Farid and I. G. Anugrah, "Implementasi CI/CD Pipeline Pada Framework Androbase Dengan Menggunakan Jenkins (Studi Kasus: PT. Andromedia)," *J. Nas. Komputasi dan Teknol. Inf.*, vol. 4, no. 6, pp. 522–527, 2021, doi: 10.32672/jnkti.v4i6.3703.

[2] A. Mustyala, "EPH- International Journal of Science And Engineering CI / CD PIPELINES IN KUBERNETES : ACCELERATING SOFTWARE," vol. 08, no. 03, pp. 1–11, 2022.

[3] J. Saltz and R. Heckman, "Exploring which agile principles students internalize when using a kanban process methodology," *J. Inf. Syst. Educ.*, vol. 31, no. 1, pp. 51–60, 2020.

[4] S. S. Pandi, P. Kumar, and R. M. Suchindhar, "Integrating Jenkins for Efficient Deployment and Orchestration across Multi-Cloud Environments," in *2023 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*, 2023, pp. 1–6. doi: 10.1109/ICSES60034.2023.10465502.

[5] Danur Wijayanto, Arizona Firdonsyah, and Faisal Dharma Adhinata, "Implementasi Continous Integration/Continous Delivery Menggunakan Process Manager 2 (Studi Kasus: SIAKAD Akademi Keperawatan Bina Insan)," *Teknika*, vol. 10, no. 3, pp. 181–188, 2021, doi: 10.34148/teknika.v10i3.400.

[6] D. Purnomo, "Model Prototyping," *JIMP-Jurnal Inform. Merdeka Pasuruan*, vol. 2, no. 2, pp. 54–61, 2017.

[7] P. Sivathapandi, H. Care, S. Corporation, V. P. Rambabu, and T. Technologies, "Advanced CI / CD Pipelines in Multi-Tenant Cloud Platforms : Strategies for Secure and Efficient Deployment," vol. 2, no. 4, pp. 212–252, 2021.

[8] D. Ghimire and S. Charters, "The Impact of Agile Development Practices on Project Outcomes," *Software*, vol. 1, no. 3, pp. 265–275, 2022, doi: 10.3390/software1030012.

[9] N. Ozkan, M. S. Gok, and B. O. Kose, "Towards a Better Understanding of Agile Mindset by Using Principles of Agile Methods," *Proc. 2020 Fed. Conf. Comput. Sci. Inf. Syst. FedCSIS 2020*, no. November, pp. 721–730, 2020, doi: 10.15439/2020F46.

[10] S. Malakar, *Agile Methodologies In-Depth: Delivering Proven Agile, SCRUM and Kanban Practices for High-Quality Business Demands (English Edition)*. Bpb Publications, 2021. [Online]. Available: https://books.google.co.id/books?id=4WMWEAAAQBAJ

[11] O. Psarov, "ENHANCING AGILE TEAM PRODUCTIVITY WITH METRICS," vol. 1, no. 113, pp. 93–99, 2024.

[12] N. Damij and T. Damij, "An Approach to Optimizing Kanban Board Workflow and Shortening the Project Management Plan," *IEEE Trans. Eng. Manag.*, vol. 71, pp. 13266–13273, 2024, doi: 10.1109/TEM.2021.3120984.

[13] J. Batskihh and T. Mynttinen, "DevOps Approach in Software Development Using Atlassian Jira Software," *Sout-Eastern Finl. Univ. Appl. Sci. Google Acad.*, 2023.

[14] C. Cowell, N. Lotz, and C. Timberlake, *Automating DevOps with GitLab CI/CD Pipelines: Build efficient CI/CD pipelines to verify, secure, and deploy your code using real-life examples*. Packt Publishing, 2023. [Online]. Available: https://books.google.co.id/books?id=X36rEAAAQBAJ

[15] P. H. P. Braga *et al.*, "Not just for programmers: How GitHub can accelerate collaborative and reproducible research in ecology and evolution," *Methods Ecol. Evol.*, vol. 14, no. 6, pp. 1364–1380, 2023, doi: 10.1111/2041-210X.14108.

[16] I. R. Onyenweaku, "A SonarQube Static Analysis of the Spectral Workbench," *Int. J. Nat. Sci. Rev.*, no. January, p. 16, 2021, doi: 10.28933/ijnsr-2020-12-0605.

[17] B. Gupta, P. Mittal, and T. Mufti, "A Review on Amazon Web Service (AWS), Microsoft Azure & Google Cloud Platform (GCP) Services," 2021, doi: 10.4108/eai.27-2-2020.2303255.

[18] R. Bullington-McGuire, A. K. Dennis, and M. Schwartz, *Docker for Developers: Develop and run your application with Docker containers using DevOps tools for continuous delivery*. Packt Publishing, 2020. [Online]. Available: https://books.google.co.id/books?id=Hif4DwAAQBAJ