ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

# SISTEM PENCADANGAN BASIS DATA ASINKRON BERBASIS TASK QUEUE MENGGUNAKAN RABBITMQ

**Ahmad Ihyaulhaq, Izzati Muhimmah**✉
Program Studi Informatika, Fakultas Teknologi Informasi, Universitas Islam Indonesia, Sleman, Indonesia
Email: izzati@uii.ac.id

DOI: https://doi.org/10.46880/jmika.Vol10No2.pp972-982

***ABSTRACT***

*Universitas Terbuka (UT) operates 39 regional offices that previously performed database backups independently, so the central office had no visibility of backup status, history, or success rates, while large synchronous backups inside the main application risked degrading service performance. This study aimed to design and implement an asynchronous backup architecture based on a RabbitMQ task queue, together with a centralized monitoring dashboard, for SIBU (Sistem Informasi Backup Universitas Terbuka). It was developed with the waterfall method (requirement, design, implementation, testing). The backend records every request as a job and publishes it to a durable queue, from which independent workers consume it in a competing-consumer pattern, dump the target PostgreSQL database, encrypt the output stream with AES-256-CBC, and upload it to MinIO object storage, while status changes reach the interface in real time through WebSocket. Black-box testing of 24 scenarios, covering job submission, parallel processing with 10 workers on a 500 MB database, status updates, failure handling, and central reporting, produced output that matched all defined criteria. The system keeps backup load outside the main API process and gives the central office one view across regional offices. Quantitative measurement of latency, throughput, and resource usage is left for future work.*

***Keywords: Asynchronous Backup, Task Queue, RabbitMQ, Message Broker, MinIO.***

**ABSTRAK**

*Universitas Terbuka (UT) memiliki 39 UT daerah yang sebelumnya menjalankan pencadangan basis data secara mandiri, sehingga UT pusat tidak memiliki visibilitas terhadap status, riwayat, maupun keberhasilan pencadangan, sementara pencadangan berskala besar yang berjalan sinkron di dalam aplikasi utama berisiko menurunkan performa layanan. Penelitian ini bertujuan merancang dan mengimplementasikan arsitektur pencadangan basis data asinkron berbasis task queue RabbitMQ disertai dasbor pemantauan terpusat pada SIBU (Sistem Informasi Backup Universitas Terbuka). Sistem dikembangkan dengan metode waterfall melalui tahap requirement, design, implementation, dan testing. Backend mencatat setiap permintaan sebagai job dan mempublikasikannya ke antrean durable, yang kemudian dikonsumsi worker independen dengan pola competing consumer; worker melakukan dump basis data PostgreSQL target, mengenkripsi aliran keluarannya dengan AES-256-CBC, dan mengunggahnya ke penyimpanan objek MinIO, sedangkan perubahan status diteruskan ke antarmuka secara real-time melalui WebSocket. Pengujian black-box terhadap 24 skenario, mencakup pengajuan job, pemrosesan paralel dengan 10 worker pada basis data 500 MB, pembaruan status, penanganan kegagalan, dan pelaporan terpusat, menghasilkan keluaran sesuai seluruh kriteria yang ditetapkan. Sistem yang dihasilkan menempatkan beban pencadangan di luar proses API utama dan menyediakan satu tampilan pencadangan bagi UT pusat. Pengukuran kuantitatif latensi, throughput, dan penggunaan sumber daya disarankan pada penelitian selanjutnya.*

***Kata Kunci: Asynchronous Backup, Task Queue, RabbitMQ, Message Broker, MinIO.***

## PENDAHULUAN

Ketersediaan sistem informasi digital menjadi aspek krusial bagi perguruan tinggi penyelenggara pendidikan jarak jauh. Terutama bagi institusi dengan skala mahasiswa besar seperti Universitas Terbuka (UT). UT merupakan perguruan tinggi negeri dengan sistem pembelajaran jarak jauh melalui 39 UT daerah di seluruh Indonesia(Lintang et al., 2026). Skala dan distribusi tersebut menjadikan basis data akademik sebagai aset yang ketersediaan dan integritasnya harus dijaga melalui mekanisme pencadangan yang dapat dikelola secara konsisten.

Pada sistem pencadangan sebelumnya, masing-masing UT daerah melakukan proses pencadangan basis data secara mandiri. Meskipun menggunakan sistem yang sama, aktivitas dan hasil pencadangan

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

tersebut belum terintegrasi dengan mekanisme pemantauan serta pelaporan secara terpusat di tingkat UT pusat. Kondisi tersebut menyebabkan UT pusat tidak memiliki visibilitas aktual terhadap status, riwayat, maupun keberhasilan pencadangan setiap unit. Selain itu, proses pencadangan basis data berukuran besar yang dijalankan secara sinkron pada aplikasi berisiko mengganggu pemrosesan permintaan lain ketika seluruh proses dijalankan (Faustino et al., 2022). Kondisi ini dapat menimbulkan kerugian signifikan apabila kegagalan pencadangan pada suatu UT daerah tidak terdeteksi sejak awal, data akademik yang hilang akan sulit direkonstruksi.

Berdasarkan hasil observasi terhadap sistem sebelumnya, permasalahan tersebut menunjukkan kebutuhan terhadap sistem yang mampu memisahkan proses pencadangan dari aplikasi. Salah satu pendekatan yang dapat digunakan adalah komunikasi asinkron menggunakan *task queue* dan *message queue*. Pola komunikasi seperti ini memungkinkan pengelolaan tugas dilakukan secara asinkron (Showkat et al., 2023). Dalam praktik pengembangan aplikasi, pekerjaan yang membutuhkan waktu pemrosesan panjang umumnya dipindahkan ke dalam mekanisme *background job* agar beban pemrosesan tidak lagi menempati proses utama aplikasi (Topalidi, 2025).

Untuk menjawab kebutuhan tersebut, UT mengembangkan SIBU (Sistem Informasi Backup Universitas Terbuka), sebuah aplikasi manajemen pencadangan akademik mahasiswa Universitas Terbuka yang memantau dan mengelola proses pencadangan dari UT daerah, serta menyediakan informasi terpusat mengenai status, riwayat, dan hasil pencadangan pada masing-masing unit.

Dalam pengembangan SIBU, mekanisme *task queue* diterapkan untuk memisahkan permintaan pencadangan dari proses utama aplikasi. Backend berperan sebagai produser yang menerima dan meneruskan permintaan pencadangan ke RabbitMQ, sedangkan *worker* independen bertindak sebagai *consumer* yang mengambil dan menjalankan tugas tersebut (Maharjan et al., 2023).

Selain pemisahan proses, SIBU menyajikan informasi pencadangan seluruh UT daerah melalui satu dasbor terpusat, sehingga informasi yang sebelumnya tersebar dapat ditampilkan dalam bentuk statistik, status, riwayat, dan rincian berdasarkan wilayah.

Berangkat dari permasalahan tersebut, penelitian ini bertujuan mengembangkan dan mengimplementasikan arsitektur pencadangan basis data asinkron pada SIBU menggunakan *task queue* berbasis RabbitMQ serta menyediakan mekanisme pemantauan pencadangan secara terpusat. Evaluasi dalam penelitian ini difokuskan pada verifikasi fungsionalitas dan alur sistem melalui *black box* testing. Pengukuran kuantitatif terhadap latensi, *throughput*, penggunaan sumber daya, dan kapasitas sistem tidak termasuk dalam ruang lingkup pengujian penelitian ini.

## KAJIAN LITERATUR

### *Task Queue*

*Task queue* merupakan mekanisme pengelolaan tugas yang memanfaatkan antrean pesan sebagai media komunikasi antara produser dan *worker*. *Task queue* melibatkan beberapa komponen, yaitu produser, broker, antrean, dan *worker*. Produser bertanggung jawab menghasilkan dan mengirimkan pekerjaan, sedangkan broker berfungsi mengatur distribusi pekerjaan menuju antrean yang sesuai. Antrean berperan sebagai penampung pekerjaan yang menunggu untuk diproses, sementara worker bertanggung jawab mengambil dan mengeksekusi pekerjaan dari antrean (Chennamsetty, 2024).

Dalam arsitektur ini, pekerjaan direpresentasikan sebagai unit yang dapat dimasukkan ke dalam antrean untuk kemudian diproses oleh *worker*. Pada mekanisme ini, permintaan pencadangan yang dikirim oleh produser tidak dieksekusi secara langsung oleh aplikasi utama, melainkan dimasukkan ke dalam antrean untuk diproses oleh *worker* secara asinkron. Mekanisme tersebut dapat digunakan untuk memisahkan pekerjaan yang membutuhkan waktu pemrosesan relatif panjang dari aplikasi yang menangani permintaan pengguna. Pendekatan ini membantu menjaga respons aplikasi sekaligus mengurangi dampak proses pencadangan terhadap performa sistem(Ćatović et al., 2022).

Prinsip model *task queue* yang digunakan pada pengembangan ini dijelaskan melalui pola *competing consumers*, yaitu sejumlah *worker* mengonsumsi pekerjaan dari satu antrean dan setiap pekerjaan hanya diproses oleh satu *worker*. Pola ini memungkinkan paralelisme diatur melalui jumlah *worker* tanpa mengubah struktur antrean (Munisif, 2022).

Pemilihan model tersebut didasarkan pada karakteristik proses pencadangan yang membutuhkan waktu pemrosesan relatif lama serta berpotensi melibatkan beberapa permintaan secara bersamaan (Gossman et al., 2024). Pemisahan proses penerimaan dan pelaksanaan pekerjaan memungkinkan pengelolaan antrean serta pengaturan jumlah pekerjaan yang diproses secara bersamaan melalui worker. SIBU menggunakan pendekatan ini untuk

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

memisahkan proses pencadangan dan enkripsi dari pemrosesan utama aplikasi. Pencadangan diperlakukan sebagai tugas yang dapat ditempatkan pada antrean dan diproses oleh worker secara independen. Model ini juga mendukung keandalan sistem: kegagalan pada satu worker dapat diisolasi tanpa menghentikan keseluruhan sistem (Ledmi et al., 2018). Di sisi lain, keterbatasan yang perlu diperhatikan adalah perlunya mekanisme acknowledgment, retry, dan idempotensi, karena pesan dapat diproses ulang pada kondisi tertentu tanpa jaminan pengiriman tepat satu kali (Patel et al., 2014).

**RabbitMQ**

RabbitMQ merupakan *message broker* berbasis AMQP (Advanced Message Queuing Protocol) yang menyediakan mekanisme *exchange, routing key, queue,* dan *consumer* untuk mendukung komunikasi asinkron (Goel, 2024). Produser mengirim pesan ke *exchange*, kemudian *exchange* meneruskan pesan ke *queue* sesuai *routing* yang ditentukan. *Queue* dapat bersifat *durable* sehingga pesan tetap tersimpan ketika server restart. *Consumer* kemudian mengambil pesan dari *queue* untuk diproses (Maharjan et al., 2023).

RabbitMQ dipilih karena menyediakan antrean *durable*, *persistent*, dan *acknowledgment* manual yang menjaga tugas pencadangan tetap tersedia saat worker tidak aktif, serta jumlah pesan antrean yang dapat digunakan untuk menskalakan worker dari nol. Perbandingan dengan platform lain menunjukkan bahwa RabbitMQ lebih sesuai untuk pola task queue dengan routing yang fleksibel, sedangkan platform seperti Kafka unggul pada aliran data ber-throughput sangat tinggi (Dobbelaere & Esmaili, 2017).

**MinIO**

MinIO merupakan *object storage* yang digunakan untuk menyimpan objek atau berkas dalam sistem terdistribusi. MinIO digunakan pada penelitian ini sebagai media penyimpanan berkas hasil pencadangan basis data. Pemisahan hasil pencadangan ke dalam media penyimpanan objek memungkinkan berkas hasil proses pencadangan tidak disimpan langsung pada penyimpanan aplikasi utama.

Penelitian Makris et al. (2022) menganalisis kinerja sistem penyimpanan termasuk MinIO pada lingkungan edge computing. Pada penelitian ini, MinIO dimanfaatkan sebagai komponen penyimpanan untuk menyimpan keluaran proses pencadangan yang telah dihasilkan oleh worker. Pemanfaatan penyimpanan objek untuk cadangan juga ditemukan pada penelitian lain, baik untuk memindahkan data cadangan dari sistem legacy ke object storage maupun untuk membangun sistem pencadangan basis data terenkripsi di atas MinIO dengan kompresi dan sinkronisasi lintas wilayah Lei (2025).

Penelitian terkait komunikasi asinkron menunjukkan bahwa message broker dapat memfasilitasi komunikasi antarkomponen secara tidak langsung dan mendukung pemisahan proses pemrosesan tugas. Ćatović et al. (2022)membahas penggunaan RabbitMQ sebagai perantara komunikasi dalam arsitektur mikroservis, khususnya untuk mendelegasikan tugas yang membutuhkan waktu dan sumber daya pemrosesan besar. Maharjan et al. (2023) membandingkan Redis, ActiveMQ Artemis, RabbitMQ, dan Kafka berdasarkan metrik latensi dan throughput. Hasil penelitian tersebut menunjukkan bahwa keunggulan setiap message queue bergantung pada metrik yang dievaluasi, dengan Redis menghasilkan latensi lebih rendah dan Kafka menunjukkan throughput lebih tinggi dalam skenario pengujian mereka. Sementara itu, Dobbelaere & Esmaili (2017) membandingkan Kafka dan RabbitMQ dari aspek fungsionalitas, kinerja, serta kesesuaian terhadap kebutuhan aplikasi. Kajian tersebut menunjukkan pentingnya menyesuaikan pemilihan teknologi antrean dengan karakteristik beban kerja dan kebutuhan sistem.

Pada aspek keandalan dan pengelolaan infrastruktur, Patel et al. (2014) mengkaji mekanisme pengiriman pesan berurutan dan tepat satu kali pada sistem antrean terdistribusi, sedangkan Ledmi et al. (2018) meninjau berbagai pendekatan toleransi kesalahan dalam sistem terdistribusi. Tran et al. (2022) membahas pendekatan dan tantangan autoscaling pada Kubernetes untuk menyesuaikan kapasitas sumber daya terhadap perubahan beban kerja. Pada aspek penyimpanan, Makris et al. (2022) mengevaluasi kinerja MinIO, BigchainDB, dan IPFS pada lingkungan edge computing menggunakan metrik kualitas layanan dan penggunaan sumber daya. Dalam lingkungan eksperimen tersebut, MinIO menunjukkan kinerja keseluruhan yang lebih baik berdasarkan metrik yang dievaluasi. Sementara itu, Lei (2025) membahas strategi pencadangan dan sinkronisasi basis data berbasis MinIO yang mengintegrasikan kompresi, enkripsi, dan sinkronisasi lintas wilayah.

Berdasarkan kajian tersebut, penelitian ini berfokus pada integrasi task queue, worker independen, enkripsi hasil pencadangan, penyimpanan objek, dan pemantauan terpusat dalam sistem SIBU untuk mendukung pengelolaan pencadangan basis data

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

di lingkungan Universitas Terbuka. Berbeda dari penelitian yang mengevaluasi komponen tertentu, seperti kinerja message broker, sistem penyimpanan, atau mekanisme autoscaling, penelitian ini menitikberatkan pada implementasi alur pencadangan secara end-to-end sesuai kebutuhan operasional institusi. Dengan demikian, kontribusi utama penelitian terletak pada integrasi dan penerapan arsitektur tersebut dalam konteks pengelolaan pencadangan lintas unit, bukan pada pengembangan algoritma baru atau pembuktian keunggulan kinerja terhadap pendekatan alternatif. Evaluasi yang dilakukan masih berfokus pada pengujian fungsional sehingga ketahanan sistem terhadap berbagai kondisi kegagalan masih perlu dikaji melalui pengujian lebih lanjut.

## METODE PENELITIAN

Dalam prosesnya, pengembangan SIBU dilakukan menggunakan metode waterfall. Metode ini melaksanakan tahapan-tahapan pengembangan perangkat lunak secara berurutan. Setiap tahapan diselesaikan sebelum tahapan berikutnya dimulai, dan hasil dari setiap tahap menjadi dasar bagi tahapan selanjutnya (Senarath, 2021). Tahapan metode waterfall terdiri dari requirement, design, implementation, dan testing akan diuraikan sebagai berikut:

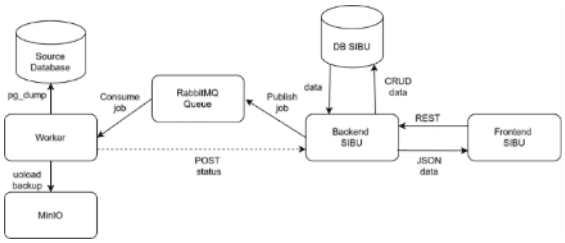
### Requirement

Tahap ini mengidentifikasi serta menganalisis kebutuhan pengguna serta menentukan prioritas fitur yang akan dikembangkan. Analisis kebutuhan pengguna dilakukan melalui observasi proses pencadangan yang berjalan di lingkungan UT serta diskusi dengan pihak pengelola sistem dan pemangku kepentingan terkait.

Analisis menunjukkan bahwa aplikasi perlu menyediakan pengelolaan dan pemantauan proses pencadangan secara terpusat, penyediaan informasi status dan riwayat pencadangan secara aktual, serta penerapan mekanisme pemrosesan asinkron untuk memisahkan proses pencadangan dari aplikasi sehingga proses pencadangan dapat berjalan secara optimal tanpa mengganggu kinerja layanan aplikasi.

### Design

Pada tahap ini dirancang arsitektur sistem, *use case diagram*, serta *activity diagram* untuk memastikan sistem dapat memenuhi kebutuhan pengguna secara optimal. Arsitektur sistem ditunjukkan pada Gambar 1.




**Gambar 1.** Arsitektur Sistem

Arsitektur sistem SIBU terdiri atas *frontend*, *backend*, basis data SIBU, RabbitMQ, *worker*, *source database*, dan MinIO. Pada bagian pencadangan *frontend* berkomunikasi dengan *backend* melalui REST API dengan pertukaran data berformat JSON, sedangkan *backend* melakukan operasi CRUD terhadap basis data SIBU untuk mengelola data aplikasi dan metadata pencadangan. Ketika pengguna mengajukan pencadangan, *backend* mempublikasikan *job* ke RabbitMQ sebagai antrean tugas. *Worker* kemudian mengonsumsi *job* tersebut dan menjalankan proses pencadangan terhadap *source database*. Hasil pencadangan diunggah ke MinIO, sedangkan status hasil proses dikirimkan kembali oleh *worker* ke *backend* melalui HTTP POST. Selanjutnya, *backend* memperbarui status dan metadata pencadangan pada basis data SIBU sehingga informasi tersebut dapat ditampilkan pada *frontend*. Arsitektur ini memisahkan eksekusi proses pencadangan dari alur pemrosesan utama *backend* melalui mekanisme *task queue*.

Alur pencadangan dirancang agar tidak ada berkas hasil dump yang utuh berada di penyimpanan sementara. Keluaran *pg_dump* dialirkan langsung ke cipher AES-256-CBC menggunakan kunci 256-bit dan initialization vector 16 byte yang dibangkitkan acak untuk setiap *job*. Berkas hasil terdiri atas header metadata tidak terenkripsi pada bagian awal yang memuat identitas job, target, waktu eksekusi, dan algoritma enkripsi lalu diikuti ciphertext. Header tersebut sengaja dibiarkan terbaca agar berkas masih dapat diidentifikasi ketika basis data SIBU hilang total, sedangkan isi cadangan tetap terlindungi.

Kunci dan IV setiap job dikirimkan worker ke backend bersama status akhir, kemudian disimpan pada rekaman job dalam bentuk terenkripsi menggunakan kunci induk (masterkey) 256-bit. Kunci induk disimpan terenkripsi dengan AES-256-GCM menggunakan kunci lingkungan dan dipecah dengan Shamir's Secret Sharing menjadi tiga bagian dengan ambang dua bagian, masing-masing dikirimkan kepada pemegang kunci yang berbeda. Dengan demikian dekripsi hasil pencadangan, baik melalui unduhan terdekripsi pada antarmuka maupun melalui prosedur pemulihan bencana, memerlukan rekonstruksi kunci induk dari

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

minimal dua pemegang kunci sehingga satu administrator tidak dapat membuka isi cadangan secara mandiri.

Pembaruan status ke antarmuka dikirim melalui WebSocket berbasis Socket.IO. Agar notifikasi tetap konsisten ketika backend dijalankan pada lebih dari satu replika, koneksi WebSocket disinkronkan antarinstance menggunakan Redis Adapter.

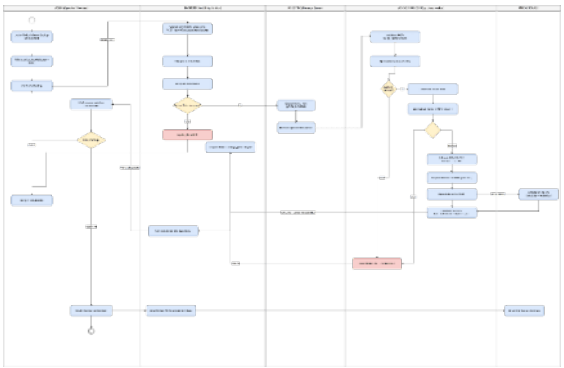



**Gambar 2.** *Use Case Diagram*

*Use case* diagram ditunjukkan pada Gambar 2 terdapat dua aktor yang berinteraksi dengan sistem, yaitu Admin UT daerah dan Admin UT pusat. Setiap aktor memiliki hak akses dan peran yang berbeda.

Admin UT Daerah dapat melakukan *login* ke dalam sistem, melakukan proses pencadangan, melihat daftar, memantau status, serta mengunduh data pencadangan yang tersedia. Selain itu, Admin UT daerah dapat melihat laporan pencadangan dan melakukan ekspor laporan ke dalam format Microsoft Excel maupun CSV.

Admin UT Pusat memiliki hak akses yang lebih luas dalam pengelolaan sistem. Admin UT Pusat dapat melakukan *login* dan menjalankan proses pencadangan, melakukan manajemen basis data, serta mengelola data *user, roles, modules, dan permission.* Selain itu, Admin UT pusat dapat mengelola peringkat, melihat laporan backup dan kinerja seluruh UT daerah, memantau *system logs*, serta melakukan pemeriksaan terhadap kondisi layanan sistem melalui fitur *cek health services*.



**Gambar 3.** *Activity Diagram*

Alur proses pencadangan yang ditunjukkan pada Gambar 3. Proses diawali oleh admin login, memilih target basis data, dan menekan tombol backup. *backend* memvalidasi permintaan tersebut, mencatat *job* dengan status PENDING, lalu mempublikasikannya ke antrean RabbitMQ apabila gagal, *job* ditandai FAILED. Status pencadangan disimpan oleh *backend* dan diteruskan ke admin secara *real-time*.

Setelah status akhir diterima, admin dapat mengunduh berkas cadangan yang tersimpan pada penyimpanan objek.

## Implementation

Tahap ini mengimplementasikan hasil desain sebelumnya ke dalam bentuk kode. Tujuan utama tahap ini adalah menghasilkan perangkat lunak yang berfungsi sesuai dengan kebutuhan yang telah ditetapkan. Implementasi mencakup pengembangan *backend*, antarmuka pengguna, serta *worker* sebagai komponen yang menjalankan proses pencadangan.

## Testing

Tahap ini memastikan bahwa perangkat lunak telah berfungsi sesuai dengan spesifikasi dan kebutuhan pengguna. Pengujian dilakukan untuk mengidentifikasi, menganalisis, dan memperbaiki kesalahan yang mungkin muncul selama proses pengembangan. Dalam penelitian ini, metode yang digunakan adalah black box testing. Pengujian yang berfokus pada evaluasi fungsionalitas sistem melalui pemeriksaan masukan dan keluaran tanpa memperhatikan struktur internal program. Metode ini digunakan untuk memverifikasi bahwa setiap fungsi menghasilkan keluaran yang sesuai dengan yang diharapkan serta untuk mengidentifikasi kesalahan pada antarmuka, model data, dan fungsionalitas sistem (Maspupah, 2024). Pendekatan ini selaras dengan karakteristik pengujian black box yang menitikberatkan pada perilaku fungsional berdasarkan

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

kebutuhan pengguna tanpa bergantung pada implementasi internal (Nidhra, 2012).

## HASIL DAN PEMBAHASAN

### Implementasi *Task Queue* dan *Worker*

Implementasi *task queue* dan *worker* pada sistem pencadangan SIBU dimulai dari alur penerimaan permintaan hingga pemrosesan pesan pada antrean. Ketika admin menekan tombol pencadangan, *backend* memvalidasi permintaan dan menyimpan catatan *job* berstatus PENDING pada basis data SIBU sebelum mempublikasikan pesan ke antrean RabbitMQ. Urutan ini membuat setiap permintaan memiliki rekam jejak meskipun pesan tidak sampai ke antrean. Jika publikasi gagal, *job* ditandai FAILED.

*Worker* berjalan sebagai proses Node.js terpisah yang mengonsumsi pesan dengan *acknowledgment* manual dan konfigurasi *prefetch(1)*, sehingga setiap instans *worker* memproses satu *job* pada satu waktu dan tingkat paralelisme ditentukan oleh jumlah instans *worker*. Untuk setiap *job*, *worker* menguji koneksi ke basis data sumber, menjalankan *dump*, dan mengalirkan keluarannya langsung ke *cipher* AES-256-CBC. Pemilihan AES-256 untuk enkripsi hasil cadangan didasarkan pada keseimbangan keamanan dan kinerja yang ditunjukkan AES dibandingkan algoritma kriptografi lainnya (Patil et al., 2016). Data terenkripsi diunggah ke MinIO secara *multipart* yang dirancang agar berkas tidak perlu dimuat utuh ke memori *worker*.

Selama proses, *worker* mengirim status melalui HTTP POST ke backend pada tahap uji koneksi, progres *dump*, dan finalisasi. *Backend* memperbarui status dan metadata pada basis data SIBU, lalu meneruskannya ke antarmuka secara *real-time* melalui WebSocket. Status *job* berjalan dari PENDING menuju IN_PROGRESS, lalu COMPLETED atau FAILED.

Pada lingkungan produksi, *worker* dijalankan di Kubernetes dengan jumlah replika awal satu. KEDA menambah instans *worker* ketika terdapat pesan dalam antrean, hingga maksimum yang ditentukan, dan Kubernetes memulai ulang instans yang berhenti. Mekanisme penambahan instans *worker* berdasarkan panjang antrean tersebut merupakan salah satu strategi *autoscaling* yang dibahas pada survei penskalaan otomatis Kubernetes (Tran et al., 2022).

Perilaku sistem pada berbagai kondisi kegagalan ditunjukkan pada Tabel 1.

**Tabel 1**. Perilaku Sistem Pada Kondisi Kegagalan

| Kondisi | Perilaku | Dampak |
|---|---|---|
| *Worker* tidak aktif sebelum mengonsumsi pesan | Pesan tetap berada pada antrean *durable* | *Job* tetap PENDING dan diproses saat *worker* aktif |
| *Worker* berhenti saat memproses (tanpa ACK) | RabbitMQ mengirim ulang pesan | *Job* diproses ulang |
| *Worker* menangkap *error* sendiri | Status FAILED dikirim ke backend satu kali, lalu pesan di-nack tanpa requeue | Status FAILED tercatat, pesan dibuang tanpa *dead-letter queue* dan tanpa *retry* |
| *Worker* berhenti atau macet tanpa sempat melapor | Belum ada deteksi otomatis | *Job* dapat tertahan pada PENDING atau IN_PROGRESS |

Pada kondisi kedua, pemrosesan ulang tidak menghasilkan berkas pencadangan ganda, melainkan penulisan ulang material enkripsi dan status *job*. Jika worker berhenti setelah status berhasil dilaporkan tetapi sebelum ACK dikirim, pesan dikirim ulang oleh broker. Pada pemrosesan ulang tersebut nama objek pada penyimpanan objek memuat job_id yang sama, berkas hasil pencadangan ditimpa dan bukan diduplikasi, sehingga pasangan baru tetap cocok dengan berkas yang tersimpan.

Meskipun pemrosesan ulang tidak menghasilkan objek cadangan yang terduplikasi, mekanisme mengharuskan pengguna memeriksa log secara manual setiap kali terjadi penumpukan *job* PENDING/IN_PROGRESS. Selain itu, karena hasil pencadangan menggunakan nama objek berdasarkan *job_id* yang sama, berkas sebelumnya akan ditimpa oleh hasil pemrosesan ulang. Apabila terjadi perubahan pada data sumber di antara kedua eksekusi, berkas yang tersimpan dapat merepresentasikan kondisi data pada waktu yang berbeda dari eksekusi sebelumnya.

Untuk kondisi terakhir, sistem hanya menyediakan indikator kesehatan pasif. Keberadaan *worker* disimpulkan secara tidak langsung dari jumlah consumer pada antrean RabbitMQ, sedangkan jumlah *job* aktif dan *job* menunggu diambil dari basis data. *job* ditandai *critical* apabila tidak terdapat *consumer* dan masih ada job berstatus PENDING, dan peringatan apabila tidak terdapat consumer dengan antrean kosong. Indikator ini bersifat informatif dan tidak mengubah status *job*. Temuan ini sejalan dengan diskusi keandalan pada sistem terdistribusi, yang

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

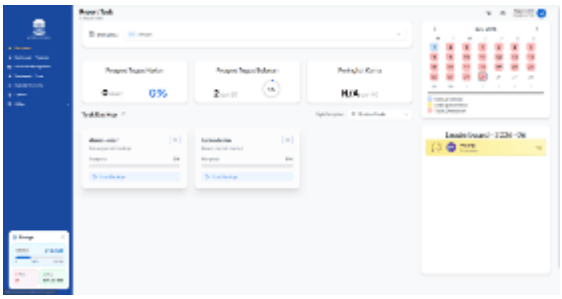
ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

menekankan bahwa penanganan kegagalan memerlukan mekanisme eksplisit seperti *retry, dead letter,* deteksi *timeout*, dan idempotensi (Ledmi et al., 2018).

Dengan demikian, sistem telah mencegah hilangnya catatan *job*. Namun sistem belum mendeteksi kegagalan tanpa laporan, belum memiliki *timeout, dead-letter queue,* maupun pembersihan objek upload yang tidak selesai. job yang sama juga belum idempoten karena setiap pengulangan tetap melakukan dump penuh. Pengembangan *automatic retry* berbatas, *dead-letter queue, timeout job, watchdog* untuk job tertahan, dan mekanisme idempotensi diusulkan sebagai penelitian lanjutan.

**Implementasi Antarmuka**

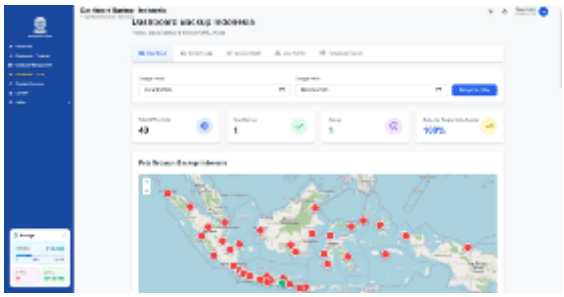
Tahap eksekusi dimulai dengan proses pengembangan kode sesuai desain arsitektur yang telah direncanakan sebelumnya. Pengembangan dilakukan secara bertahap pada masing-masing komponen.



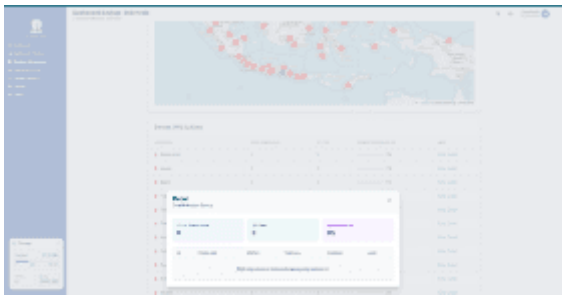
**Gambar 4.** Tampilan Dashboard Utama

Pengembangan pertama dilakukan pada bagian *frontend* mengikuti desain yang sudah diberikan. Halaman pertama yang dikembangkan merupakan halaman dashboard utama dimana proses pencadangan dilakukan. Pada halaman ini selain melakukan pencadangan terdapat ringkasan singkat dari pencadangan yang harus dilakukan pada hari itu untuk setiap UT daerah, kalender ringkasan dan peringkat seluruh UT daerah. Seperti yang ditunjukkan pada Gambar 4.

Halaman berikutnya gambar 5 dan gambar 6 merupakan dashboard pemantauan yang digunakan oleh administrator UT pusat. Dashboard ini berfungsi untuk memantau kondisi sistem secara keseluruhan dengan menampilkan informasi berupa *application log*, status kesehatan setiap layanan, aktivitas pengguna, serta ringkasan proses pencadangan yang dilakukan oleh seluruh UT daerah.



**Gambar 5.** Antarmuka Dashboard Admin Pusat



**Gambar 6.** Detail Setiap UT

Sebagaimana ditunjukkan pada Gambar 5, halaman dashboard menyediakan fitur penyaringan data berdasarkan rentang tanggal. Selain itu, ditampilkan ringkasan statistik pencadangan secara nasional serta peta Indonesia yang menampilkan lokasi setiap UT daerah dalam bentuk marker. Setiap marker dapat menampilkan informasi singkat mengenai kondisi pencadangan ketika di *hover*. Pada bagian bawah halaman, tersedia tabel yang memuat rincian aktivitas pencadangan dari masing-masing UT daerah. Setiap entri pada tabel juga dilengkapi dengan modal dialog yang menampilkan informasi lebih rinci mengenai proses pencadangan yang telah dilakukan.

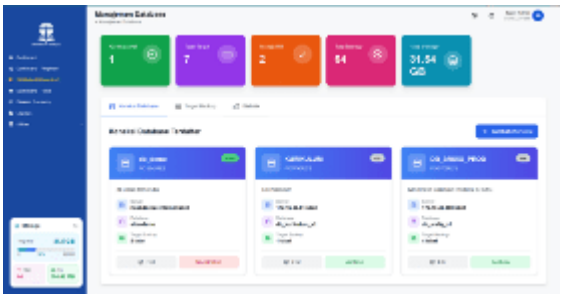


**Gambar 7.** Tampilan User Activity

Pada tab *user activity* di halaman yang sama ditampilkan informasi aktivitas pencadangan yang dilakukan oleh pengguna. Informasi yang disajikan meliputi jumlah pengguna aktif, jumlah keseluruhan pengguna, pengguna dengan frekuensi pencadangan tertinggi, serta riwayat proses pencadangan yang dilakukan beserta statusnya. Tampilan halaman *user activity* ditunjukkan pada Gambar 7.

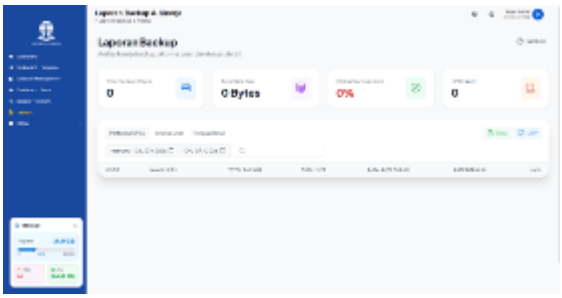
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)



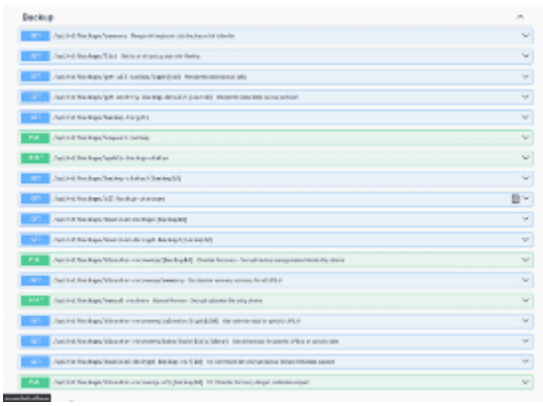
**Gambar 8.** Tampilan Halaman Manajemen Database

Halaman manajemen basis data dikembangkan untuk memudahkan admin UT pusat dalam mengelola koneksi basis data yang menjadi target proses pencadangan. Sebagaimana ditunjukkan pada Gambar 8, halaman ini menampilkan ringkasan informasi mengenai jumlah koneksi basis data yang aktif, jumlah target pencadangan yang terdaftar, jumlah target yang sedang aktif, total pencadangan yang telah dilakukan, serta kapasitas penyimpanan yang digunakan. Halaman ini juga menyediakan beberapa tab untuk mengelola koneksi basis data, target pencadangan, dan melihat statistik. Pada bagian koneksi *database*, setiap basis data ditampilkan dalam bentuk kartu yang memuat informasi mengenai basis data tersebut.



**Gambar 9.** Tampilan Halaman Laporan Backup

Halaman laporan *backup*, ditunjukkan pada Gambar 9, menyediakan informasi mengenai hasil dan performa proses pencadangan yang telah dilakukan. Halaman ini menampilkan ringkasan berupa total pemeriksaan pencadangan, total ukuran data yang telah dicadangkan, persentase tingkat keberhasilan pencadangan, serta jumlah UT daerah yang aktif. Admin dapat melakukan penyaringan laporan sesuai rentang waktu tertentu untuk menampilkan data yang diinginkan. Informasi hasil pencadangan kemudian disajikan dalam bentuk tabel yang memuat data setiap UT daerah, termasuk jumlah pencadangan, ukuran data, rata-rata durasi proses, dan tingkat keberhasilan. Laporan juga dapat diekspor ke dalam format Excel dan CSV.



**Gambar 10.** Daftar API Backup

Setelah mengimplementasikan tampilan, dilanjut dengan pengembangan API dan *worker* untuk mendukung proses pencadangan. Fungsionalitas yang diimplementasikan meliputi alur proses pencadangan, pembaruan dan pemantauan status pencadangan, penyajian ringkasan serta riwayat pencadangan serta pengunduhan berkas cadangan dalam bentuk terenkripsi maupun terdeskripsi. Sistem juga menyediakan API untuk pengelolaan target pencadangan, rekapitulasi data pencadangan berdasarkan wilayah dan periode tertentu, serta penyampaian informasi progres pencadangan kepada antarmuka pengguna. Gambar 10 menunjukkan daftar API yang telah diimplementasikan.

**Pengujian**

Pada tahap ini, pengujian dilakukan menggunakan pendekatan *black box testing* untuk memverifikasi fungsionalitas sistem telah berjalan sesuai dengan kebutuhan pengguna. Pengujian dilakukan secara langsung melalui antarmuka aplikasi dengan berfokus pada validasi masukan, keluaran, serta respons sistem tanpa memperhatikan implementasi kode program. Pendekatan serupa digunakan pada penelitian (Sanjaya et al., 2025), yang menerapkan *black box* testing dengan mengevaluasi fungsionalitas sistem berdasarkan masukan dan keluaran yang diharapkan.

Pengujian dilakukan pada lingkungan Windows 11. SIBU dijalankan menggunakan Docker Compose dan scaling worker menggunakan Kubernetes, dengan PostgreSQL versi 18, RabbitMQ versi 3.11, dan MinIO latest. Basis data yang digunakan sebagai sumber pencadangan memiliki ukuran ±500MB, sedangkan proses pencadangan dijalankan dengan 10 instans worker. Pengujian mencakup 24 skenario yang dikelompokkan berdasarkan fungsionalitas antarmuka dan alur pencadangan. Kriteria keberhasilan berupa keluaran sesuai kriteria yang ditetapkan pada tiap skenario.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Pengujian difokuskan pada fungsi yang berkaitan langsung dengan proses pencadangan asinkron dan pemantauan pencadangan. Seluruh pengujian dilakukan untuk memverifikasi alur sistem secara *end-to-end* telah berjalan dengan baik sehingga setiap perubahan status maupun data dapat ditampilkan secara akurat pada antarmuka pengguna.

Pengujian fungsionalitas dasbor UT daerah ditunjukkan pada **Tabel 2**. Kesepuluh skenario memeriksa pengajuan pencadangan, pemrosesan pencadangan paralel, pengunduhan hasil, pemilihan UT daerah, pembaruan progres, penanganan kegagalan pencadangan, serta ringkasan harian/bulanan dan papan peringkat.

**Tabel 2**. Pengujian Antarmuka Halaman Dashboard Utama

| Fitur | Pengujian | Harapan | Hasil |
|---|---|---|---|
| Tombol mulai Pencadangan | Ketuk tombol pencadangan pada kartu Pencadangan | Aplikasi memproses permintaan Pencadangan sesuai kartu dan memberikan *feedback* | Sesuai |
| Pencadangan Paralel | Ketuk tombol pencadangan saat pencadangan lain berlangsung | Aplikasi memproses permintaan Pencadangan sesuai batas dan antrean | Sesuai |
| Tombol mulai *download* | Ketuk tombol completed | Aplikasi mendownload file hasil *backup* ke *local user* | Sesuai |
| Pemilihan UT daerah | Pilih UT daerah yang valid dari menu dropdown | Data dasbor diperbarui sesuai dengan UT daerah yang dipilih | Sesuai |
| *Update* progress Pencadangan | Jalankan proses pencadangan | Bilah kemajuan diperbarui sesuai dengan status pencadangan | Sesuai |
| Penanganan kegagalan pencadangan | Simulasi pencadangan gagal | Pesan dan status gagal ditampilkan kepada pengguna | Sesuai |
| Perhitungan ringkasan harian, bulanan serta peringkat | Melakukan pencadangan dan melihat hasil yang ditampilkan | Ringkasan diperbarui setelah backup berhasil | Sesuai |
| Menampilkan *leaderboard* | Membuka halaman dashboard | Halaman menampilkan peringkat sesuai data yang tersedia | Sesuai |
| Menampilkan Kalender | Membuka halaman dashboard | Kalender menampilkan tanggal dan hasil pencadangan tiap tanggal pada bulan ini | Sesuai |
| Pembaruan papan peringkat | Selesaikan tugas pencadangan | Skor dan peringkat pengguna diperbarui dengan benar | Sesuai |

Pengujian fungsionalitas dasbor UT pusat ditunjukkan pada Tabel 3. Keenam skenario memeriksa kartu ringkasan statistik, penyaringan berdasarkan rentang tanggal, peta dan marker UT daerah, tabel ringkasan tiap UT daerah, serta modal detail tiap UT daerah.

**Tabel 3**. Pengujian Antarmuka Halaman Dasbor UT Pusat

| Fitur | Pengujian | Harapan | Hasil |
|---|---|---|---|
| kartu ringkasan statistik | Buka halaman dashboard | Total UPBJJ, Total Cadangan, Selesai, dan Tingkat Keberhasilan ditampilkan dengan benar | Sesuai |
| Filter tanggal | Set filter ke periode pencadangan yang tersedia | Statistik, tabel dan peta diperbarui dengan benar | Sesuai |
| Peta UT daerah | Buka halaman dashboard | Peta Indonesia marker UT daerah | Sesuai |
| Marker UT daerah | Hover di atas marker pada peta | Detail tiap marker ditampilkan | Sesuai |
| Tabel ringkasan tiap UT daerah | Buka halaman dasbor | Semua UT daerah ditampilkan sesuai data | Sesuai |
| Modal detail tiap UT daerah | Tekan detail pada tabel | Modal ditampilkan sesuai data dan UT daerah yang dipilih | Sesuai |

Pengujian pemulihan hasil pencadangan ditunjukkan pada Tabel 4. Skenario pertama memeriksa kesesuaian isi basis data hasil pemulihan dengan sumber melalui jumlah baris dan hash isi tabel target, sedangkan skenario kedua memeriksa kegagalan dekripsi ketika kunci yang digunakan tidak sesuai.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

**Tabel 4**. Pengujian Pemulihan *Backup*

| Fitur | Pengujian | Harapan | Hasil |
|---|---|---|---|
| Pemulihan basis data menggunakan data yang dicadangkan | Cadangkan, unduh, dekripsi, lalu pemulihan ke basis data kosong | Jumlah baris dan hash isi tabel target identik dengan sumber | Sesuai |
| Dekripsi basis data | Dekripsi dengan kunci yang salah | Restore gagal | Sesuai |

Pengujian integrasi sistem ditunjukkan pada Tabel 5. Skenario memeriksa pengiriman tugas, pemrosesan oleh worker, penempatan tugas pada antrean, pembaruan status, serta tiga kondisi kegagalan, yaitu kegagalan yang dilaporkan worker, worker tidak aktif sebelum mengonsumsi pesan, dan worker berhenti di tengah proses pemrosesan.

**Tabel 5**. Pengujian Integrasi Sistem

| Fitur | Pengujian | Harapan | Hasil |
|---|---|---|---|
| Pengiriman tugas | Ajukan pencadangan melalui UI | Job tercatat dan dikirim ke RabbitMQ | Sesuai |
| Pemrosesan worker | Amati *log job* yang telah dikirim | Worker mengambil dan memproses *job* | Sesuai |
| Antrean tugas | Ajukan beberapa *job* saat *worker* sedang bekerja | *Job* yang belum diproses berada dalam antrean | Sesuai |
| Pembaruan status | Amati log proses hingga selesai | Status berubah dan ditampilkan pada dashboard | Sesuai |
| Kegagalan proses | Trigger kondisi kegagalan | Status FAILED dan informasi kegagalan disimpan | Sesuai |
| Worker tidak aktif | *Worker* dimatikan, *job* diajukan, lalu *worker* dinyalakan | *Job* PENDING lalu diproses hingga SUCCESS | Sesuai |
| *Worker* berhenti saat memproses (tanpa ACK) | Hentikan *worker* paksa di tengah proses *dump* | Pesan dikirim ulang oleh broker dan *job* diproses ulang | Sesuai |

Berdasarkan hasil pengujian terhadap seluruh skenario, disimpulkan sistem mampu menghasilkan keluaran yang sesuai dengan ekspektasi dan kriteria pengujian yang telah ditetapkan. Oleh karena itu, fitur-fitur utama pada sistem dapat dinyatakan berjalan secara fungsional dan telah memenuhi spesifikasi kebutuhan yang dirancang.

Dari sisi keandalan, kombinasi antrean *durable,* pesan *persistent*, dan *acknowledgment* manual menjaga agar job yang sudah berada di antrean tidak hilang saat *worker* tidak aktif atau berhenti di tengah proses. *Job* dapat diproses ulang. Pada sisi produser, mesin antrean telah menggunakan *publisher confirms*, sehingga pesan yang gagal sampai ke antrean terdeteksi dan status *job* langsung ditandai FAILED, tidak lagi tertahan pada PENDING. Kegagalan yang dilaporkan *worker* tercatat, sedangkan kegagalan tanpa laporan yang masih bergantung pada pemeriksaan manual kini hanya mencakup kondisi *worker* macet, ketika pesan yang telah terkonfirmasi masuk ke antrean tetapi tidak kunjung diproses, sehingga *job* tertahan pada status PENDING atau IN_PROGRESS.

## KESIMPULAN

Penelitian ini menghasilkan arsitektur pencadangan basis data asinkron pada SIBU yang memisahkan eksekusi pencadangan dari aplikasi utama melalui *task queue* RabbitMQ dan *worker* independen, serta dasbor pemantauan terpusat bagi UT pusat. Hasil pengujian *black box* menunjukkan bahwa fungsi yang diuji, mencakup pengajuan tugas, antrean, pembaruan status, pelaporan kegagalan, dan pelaporan terpusat, menghasilkan keluaran sesuai kebutuhan yang ditetapkan. Dengan demikian, tujuan penelitian untuk mengembangkan dan mengimplementasikan arsitektur tersebut serta pemantauan terpusat telah terpenuhi secara fungsional. Arsitektur ini dirancang agar beban pencadangan berada di luar proses API utama; namun karena pengujian bersifat fungsional, penelitian ini belum dapat menyimpulkan dampaknya terhadap latensi maupun *throughput*.

Sistem mendeteksi dan mencatat kegagalan yang dilaporkan oleh *worker*, tetapi belum mendeteksi *job* yang tertahan akibat *worker* berhenti tanpa melapor, belum memiliki *retry* maupun *dead-letter queue*, dan dapat memproses ulang *job* tanpa jaminan idempotensi. Pencadangan yang bersifat logis juga tidak menyediakan pemulihan pada titik waktu tertentu.

Penelitian selanjutnya dapat mengembangkan pengujian beban untuk mengukur secara kuantitatif pengaruh arsitektur asinkron terhadap responsivitas dan kapasitas aplikasi utama selama proses pencadangan. Pendekatan ini dapat memanfaatkan alat seperti Apache JMeter dan k6.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

**DAFTAR PUSTAKA**

Ćatović, A., Buzađija, N., & Lemes, S. (2022). Microservice development using RabbitMQ message broker. *Science, Engineering and Technology*, *2*(1), 30–37. https://doi.org/10.54327/set2022/v2.i1.19

Chennamsetty, C. S. (2024). Real-Time Notifications and Event-Driven Architectures: Scaling Proactive Communication for Customer Retention. *International Journal of Advanced Research in Computer Science & Technology (IJARCST)*, *7*(1), 9686–9691. https://doi.org/10.15662/IJARCST.2024.0701003

Dobbelaere, P., & Esmaili, K. S. (2017). *Kafka versus RabbitMQ*. http://arxiv.org/abs/1709.00333

Faustino, D., Gonçalves, N., Portela, M., & Silva, A. R. (2022). *Stepwise Migration of a Monolith to a Microservices Architecture: Performance and Migration Effort Evaluation*. http://arxiv.org/abs/2201.07226

Goel, R. (2024). Evaluating Message Brokers: Performance, Scalability, and Suitability for Distributed Applications. *American Journal of Computer Architecture*, *11*(5), 62–65. https://doi.org/10.5923/j.ajca.20241105.02

Gossman, M. J., Nicolae, B., & Calhoun, J. C. (2024). Scalable I/O aggregation for asynchronous multi-level checkpointing. *Future Generation Computer Systems*, *160*, 420–432. https://doi.org/10.1016/J.FUTURE.2024.06.003

Ledmi, A., Bendjenna, H., & Hemam, S. M. (2018). *Fault Tolerance in Distributed Systems: A Survey*. 2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS). https://doi.org/10.1109/PAIS.2018.8598484

Lei, J. (2025). MinIO-Based Efficient Database Backup and Synchronization Strategy: Compressed and Encrypted Storage with Cross-Region Real-Time Synchronization. *Applied Science and Innovative Research*, *9*(1), p136. https://doi.org/10.22158/asir.v9n1p136

Lintang, Y., Andrian Putro, C., Afriyanti, R., Sadid Parassa, H., Sandi, F. A., & Priyanto, A. (2026). Student Characteristics and Learning Support Design in Open and Distance Higher Education: Evidence from Universitas Terbuka. *Media Komunikasi FPIPS*, *25*(1), 9–27. https://doi.org/10.23887/mkfis.v25i1.107174

Maharjan, R., Chy, M. S. H., Arju, M. A., & Cerny, T. (2023). Benchmarking Message Queues. *Telecom*, *4*(2), 298–312. https://doi.org/10.3390/telecom4020018

Makris, A., Kontopoulos, I., Psomakelis, E., Xyalis, S. N., Theodoropoulos, T., & Tserpes, K. (2022). Performance Analysis of Storage Systems in Edge Computing Infrastructures. *Applied Sciences (Switzerland)*, *12*(17). https://doi.org/10.3390/app12178923

Maspupah, A. (2024). Literature Review: Advantages and Disadvantages of Black Box and White Box Testing Methods. *Jurnal Techno Nusa Mandiri*, *21*(2), 151–162. https://doi.org/10.33480/techno.v21i2.5776

Munisif, B. R. (2022). Asynchronous Messaging Architectures for Distributed Job Processing Systems. *International Journal of Computer Technology and Electronics Communication*, *05*(05). https://doi.org/10.15680/IJCTECE.2022.0505009

Nidhra, S. (2012). Black Box and White Box Testing Techniques - A Literature Review. *International Journal of Embedded Systems and Applications*, *2*(2), 29–50. https://doi.org/10.5121/ijesa.2012.2204

Patel, D., Khasib, F., Sadooghi, I., & Raicu, I. (2014). *Towards In-Order and Exactly Once Delivery using Hierarchical Distributed Message Queues*.

Patil, P., Narayankar, P., Narayan, D. G., & Meena, S. M. (2016). A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish. *Procedia Computer Science*, *78*, 617–624. https://doi.org/10.1016/j.procs.2016.02.108

Sanjaya, I. G. W., Kartika, L. G. S., & Utama, P. K. L. (2025). Perancangan E-Sukerti (Surat Keterangan Elektronik) Desa Baturiti Berbasis Web Berbasis Black Box Testing. *METHOMIKA Jurnal Manajemen Informatika Dan Komputerisasi Akuntansi*, *9*(2), 264–274. https://doi.org/10.46880/jmika.vol9no2.pp264-274

Senarath, U. S. (2021). *Waterfall Methodology, Prototyping and Agile Development*. https://doi.org/10.13140/RG.2.2.17918.72001

Showkat, M., Chy, H., Ashfakur, M., Arju, R., Tella, M., & Cerny, T. (2023). Comparative Evaluation of Java Virtual Machine-Based Message Queue Services: A Study on Kafka, Artemis, Pulsar, and RocketMQ. *Electronics*, (Article 4792). https://doi.org/10.3390/electronics12234792

Topalidi, A. (2025). Asynchronous processes and message queues in Ruby applications: efficiency analysis of Sidekiq and RabbitMQ. *International Journal on Science and Technology*, *16*(4). https://doi.org/10.71097/IJSAT.v16.i4.9271

Tran, M.-N., Vu, D.-D., & Kim, Y. (2022). *A Survey of Autoscaling in Kubernetes*. https://doi.org/10.1109/ICUFN55119.2022.9829572

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License. 