BAB V
KESIMPULAN

5.1. Kesimpulan

Berdasarkan proses perancangan, implementasi, dan pengujian yang telah dilakukan, Arsitektur sistem deteksi hoaks berhasil dirancang menggunakan pendekatan berbasis *service* dengan memisahkan proses penerimaan request, antrean pekerjaan, dan pemrosesan *pipeline* komputasi. *Gateway* menerima citra melalui *endpoint POST /predict/async*, memasukkan *job* ke *Redis Queue*, lalu mengembalikan *job_id* tanpa menunggu seluruh proses selesai. *Worker* kemudian mengambil pekerjaan dari antrean untuk menjalankan pra-pengolahan citra, *OCR*, dan klasifikasi *IndoBERT*, sedangkan *client* memeriksa status serta mengambil hasil melalui *endpoint GET /predict/async/{job_id}*. Mekanisme ini menghasilkan pemrosesan asinkron dan temporal *decoupling*, dengan *Redis Queue* sebagai *buffer* dan beberapa *worker replica* untuk memproses pekerjaan secara paralel. Namun, waktu respons *API asinkron* tidak merepresentasikan waktu penyelesaian prediksi secara menyeluruh karena waktu tunggu antrean dan pemrosesan *worker* berada di luar respons awal.

Horizontal Pod Autoscaler berhasil diterapkan pada *Deployment* yang menjalankan beban komputasi menggunakan metrik utilisasi *CPU*. *HPA* dikonfigurasi dengan target *CPU*, jumlah minimum dan maksimum replica, kebijakan *scale-up* tanpa *stabilization window*, serta *scale-down stabilization window* selama 120 detik. Ketika beban meningkat dari *baseline* ke *ramp-up* dan *stress*, jumlah *pod* pada lingkungan lokal meningkat dari 20 menjadi 23 dan 25 *pod*, sedangkan pada lingkungan server meningkat dari 16 menjadi 20 *pod*. Pada skenario *soak*, jumlah *pod* menurun menjadi 22 pada lingkungan lokal dan 18 pada lingkungan server setelah intensitas beban berkurang. Hasil tersebut menunjukkan bahwa *HPA* dapat menyesuaikan kapasitas layanan terhadap perubahan beban, dengan pembuktian yang seharusnya didasarkan pada metrik *CPU* dan jumlah replica pada *Deployment target*.

Hasil pengujian menunjukkan bahwa arsitektur berbasis *service* menghasilkan *aggregate HTTP throughput* lebih tinggi dan P95 *API latency* lebih rendah dibandingkan arsitektur monolitik. Pada *stress testing*, *service* lokal dan server menghasilkan 147 *req/s* dan 177 *req/s*, sedangkan monolitik lokal dan server hanya mencapai 2,67 req/s dan 0,689 req/s. P95 *latency service* sebesar 44,7 ms dan 67,3 ms merepresentasikan waktu respons *API* untuk pengiriman pekerjaan dan *polling* status, sedangkan P95 monolitik mencakup seluruh *pipeline* sinkron yang mencapai sekitar 10 detik. Uji *Wilcoxon* menghasilkan $W = 0$ dan *p-value* = 0,0078 pada *throughput* dan P95 *latency*, sehingga terdapat perbedaan kinerja yang signifikan. Meskipun demikian, nilai 177 *req/s* tidak dapat diartikan sebagai 177 prediksi selesai per detik karena mencakup *request polling*, sehingga peningkatan

kinerja dipahami sebagai dampak gabungan dari arsitektur asinkron, *Redis Queue*, orkestrasi *Kubernetes*, jumlah replica, dan *HPA*.

5.2. Saran

Berdasarkan hasil dan keterbatasan penelitian, beberapa saran yang dapat diberikan untuk penelitian selanjutnya adalah sebagai berikut:

1. Penelitian selanjutnya disarankan membandingkan arsitektur *service* yang sama dalam dua kondisi, yaitu *HPA* aktif dan *HPA* nonaktif. Perbandingan tersebut diperlukan agar pengaruh *HPA* dapat diisolasi dari pengaruh perubahan arsitektur dan pemrosesan asinkron. Dengan demikian, peningkatan *throughput*, penurunan *latency*, dan perubahan penggunaan sumber daya dapat dikaitkan secara lebih langsung dengan mekanisme *autoscaling*.
2. Pengujian sebaiknya dilakukan melalui beberapa kali pengulangan untuk setiap kombinasi lingkungan, arsitektur, dan skenario beban. Data dari setiap pengulangan dapat digunakan sebagai pasangan observasi dalam pengujian statistik sehingga kesimpulan tidak hanya bergantung pada nilai ringkasan dari satu kali pengujian. Jumlah sampel yang lebih besar juga memungkinkan perhitungan *effect size* dan interval kepercayaan.
3. Penelitian selanjutnya dapat membandingkan *HPA* standar dengan pendekatan *autoscaling* lain, seperti *HPA* berbasis *multi-metric*, *Vertical Pod Autoscaler*, *Kubernetes Event-Driven Autoscaling*, atau algoritma prediktif. Perbandingan tersebut dapat digunakan untuk menentukan mekanisme *scaling* yang paling sesuai bagi *pipeline OCR* dan *NLP*.
4. Antarmuka web yang telah dikembangkan dapat dilanjutkan dengan pengujian *usability*, keamanan, dan kompatibilitas perangkat. Pengujian keamanan dapat mencakup validasi *JWT*, masa berlaku token, pembatasan ukuran berkas, validasi tipe citra, *rate limiting*, serta perlindungan *endpoint API*. Namun, pengujian tersebut perlu ditempatkan sebagai penelitian terpisah karena penelitian saat ini berfokus pada *performance testing*.
5. Penelitian selanjutnya disarankan menggunakan server yang dilengkapi *GPU* untuk menjalankan proses *OCR* dan inferensi model *IndoBERT*. Pada penelitian ini, seluruh proses komputasi masih dijalankan menggunakan *CPU* sehingga waktu pemrosesan dipengaruhi oleh keterbatasan sumber daya komputasi. Penggunaan *GPU* diharapkan dapat mempercepat proses inferensi, menurunkan *latency*, meningkatkan *throughput*, serta mengurangi jumlah replica yang diperlukan untuk menangani beban yang sama. Selain itu, hasil pengujian pada lingkungan *GPU* dapat dibandingkan dengan lingkungan *CPU* untuk mengetahui pengaruh jenis akselerator terhadap efektivitas *Horizontal Pod Autoscaler* dan efisiensi sumber daya sistem.