Backend Architecture Analysis Report

Student: Ahmed Al-Hannawi

Task: Architecture Decision for Company A, B, C

Date: 2025-10-01

Executive Summary

This report recommends optimal architecture patterns for three companies with distinct workload characteristics:

• Company A (Real-time Chat): Event-loop / single-threaded processes clustered per core + offload for CPU-bound tasks.
• Company B (Image Processing): Controlled worker pool (multi-process) tuned to CPU/GPU and memory; job queue.
• Company C (E-commerce API Gateway): Multiplexing-focused async gateway (HTTP/2 / connection pools) with auto-scaling.

Each recommendation includes technical justification, implementation sketch, trade-offs, rejected alternatives, and performance estimates with calculations.

Company A — Real-time Chat Platform

Requirements Summary

• 10,000 concurrent WebSocket connections.
• Small messages (<1KB), low latency target (<50ms), mostly I/O-bound (80%).
• 8-core server, DB latency 10–200ms, memory ~500MB app footprint.

Recommendation

• Use an async event-loop model (single-threaded per process, e.g., Node.js or asyncio) AND run a cluster of processes (one process per CPU core).
• Offload CPU-heavy tasks (20% of work) to a worker pool (threadpool or separate processes).
• Use async DB drivers, connection pooling, and Redis (or Kafka) pub/sub for inter-process message distribution.

Technical Justification

• The workload is predominantly I/O-bound: the event-loop excels at keeping a single thread utilized while awaiting I/O.

• Multiple event-loop processes (one per core) use available cores without forcing multithreading.
• Offloading CPU tasks prevents blocking and latency spikes.

Trade-offs Accepted

• Memory duplication: Each process holds ~0.5GB; 8 processes ≈ 4GB.
• Added complexity: worker pool + clustering.

Rejected Alternatives

• Multi-threaded blocking server: threads blocked on I/O waste resources.
• Single-process event-loop: doesn't utilize 8 cores.

Performance Estimates

Expected message rate ~250 msg/s. With offloading, target p95 latency <50ms is achievable.

Company B — Image Processing Service

Requirements Summary

• Avg image 5MB, 2–10s processing, ~50 concurrent uploads.
• 16-core server with GPU acceleration, ~2GB per image memory.
• 90% CPU-bound, independent tasks.

Recommendation

• Multi-process worker pool with job queue (e.g., RabbitMQ/Redis).
• Concurrency limited by cores, memory, and GPU slots.
• Autoscale when queue length grows.

Technical Justification

• CPU/GPU heavy work needs isolated workers.
• Memory footprint large, concurrency must be capped.
• GPUs best utilized via isolated workers.

Trade-offs Accepted

• Need large-memory instance or cluster.
• Job queue complexity + idempotency handling.

Rejected Alternatives

• Single-threaded async: blocks on CPU.
• Multiplexing: doesn't help with CPU-bound tasks.

Performance Estimates

16 workers, avg 6s per job → throughput ≈ 2.7 images/sec. Memory constraint dominates.

Company C — E-commerce API Gateway

Requirements Summary

• 1,000 RPS, fan-out to 20+ services.
• 70% I/O-bound, 30% CPU.
• 4-core server, latency 100–500ms.
• Must handle 5x spikes.

Recommendation

• Async multiplexing gateway (HTTP/2 / persistent pools).
• Parallel fan-out, caching, circuit breakers.
• Autoscaling required.

Technical Justification

• Multiplexing reduces connection churn.
• Async parallel calls overlap I/O waits.
• Autoscaling mitigates 5x spikes.

Trade-offs Accepted

• Error handling complexity.
• Requires observability + traffic shaping.

Rejected Alternatives

• Multi-threaded gateway: high overhead.
• Pure single-threaded: high connection churn.

Performance Estimates

Normal outstanding calls ≈ 1,200. On 4 cores, must scale horizontally for spikes.