Architecture solutions

Here's a NestJS backend API that handles image/video uploads, processes them via a queue, and analyzes images to get an exposure time using a mock OpenAI API.

To handle 100 / 1,000 / 10,000+ concurrent events efficiently, the system needs to scale dynamically. Below is a scalable architecture diagram and breakdown of how the system adapts as event volume increases.

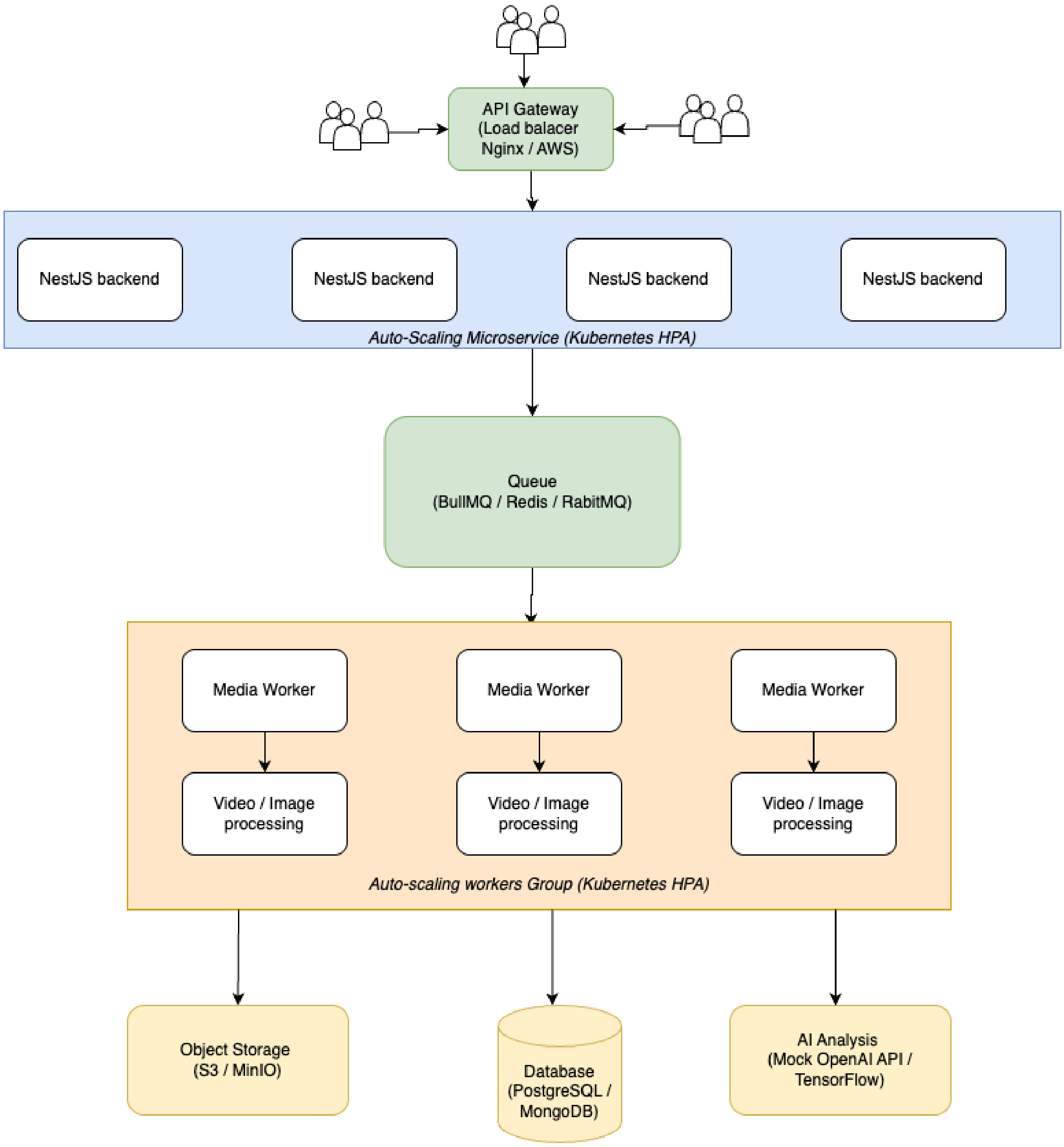

Image: an architecture diagram for a scalable system.

API Gateway (NGINX / AWS API Gateway)

- Load balances requests to backend services.
- Manages authentication and rate limiting.

Microservices (NestJS)

- Handles uploads and queues tasks for processing.
- Scales horizontally based on traffic.
- Runs on Kubernetes clusters.

Queue System (BullMQ / Redis / RabitMQ)

- Decouples processing from request handling.
- Ensures reliability under high loads.

Object Storage (S3 / MinIO)

- Stores large video/image files efficiently.

Media Workers : Image | Video Processing (FFmpeg Workers)

- Extracts frames at 10s intervals.
- Deploys auto-scaled processing workers.

AI Analysis (Mock OpenAI API / TensorFlow)

- Analyzes images for brand exposure detection.
- Runs on Kubernetes clusters.

Database (PostgreSQL / MongoDB / DynamoDB)

- Stores event metadata and exposure results.
- Uses caching (Redis) for faster lookups.

Monitoring & Scaling (Kubernetes HPA)

- Monitors CPU, memory, queue length.
- Auto-scales workers based on event volume.

Event Volume	**System Scaling Approach**
100 events	Single instance, small Redis queue, minimal processing workers.
1000 events	Multiple backend replicas, auto-scaling workers, DB read replicas.
10000+ events	Multi-region support, sharded DB, high-throughput object storage, GPU instances for AI analysis.