Nebula GPU Cloud – Architecture Document

Author: Rares Migea

1. Overview

Nebula GPU Cloud is a simplified AI PaaS platform that simulates GPU-based inference and provides full-stack features including:

- API key management with rate limiting
- Model deployment and selection
- Distributed inference via a task queue
- Usage metrics and monitoring
- OpenAI-compatible chat endpoints
- Web frontend dashboard for interactive testing

The system is built with Node.js, Redis, BullMQ, and React.

2. System Design & Component Interaction

Backend Components:

- **Express API**: Handles HTTP requests, routes for `/v1/chat/completions`, `/apikeys`, `/models`, `/metrics`, `/health`.
- **API Key Service**: Creates, stores, and validates API keys in-memory. Integrates with rate limiting and usage tracking.

- **Rate Limiting Middleware**: Uses Redis counters to enforce per-key request limits.
- **Telemetry Middleware**: Records request count and last-used timestamp in Redis.
- **Inference Queue (BullMQ)**: Manages asynchronous inference tasks across workers.
- **Workers**: Simulate GPU processing with variable delays; return model output to the queue.
- **Redis**: Central store for queue, rate limiting, and usage metrics.

Frontend Components:

- **React SPA** with React Router for multiple pages:
  - Dashboard (health + usage metrics)
  - API Keys management
  - Chat interface (run inferences)
  - Model deployment interface
- **Axios**: Communicates with backend API endpoints.

Component Interaction Diagram:

```
Frontend
 ├─ Dashboard <---> /metrics
 ├─ API Keys <---> /apikeys
 └─ Chat <---> /v1/chat/completions
                   |
                   └─ BullMQ Queue --> Worker(s) --> Redis (store
results)
```

---

3. Distributed Architecture Approach

- **Task Queue (BullMQ + Redis)** simulates distributed GPU workers.

- Each worker subscribes to the same `inference` queue.

- Jobs are asynchronous and independent; multiple workers can be added horizontally.

- **Redis** centralizes queue state and metrics, allowing multiple backend instances to share tasks.

- **Streaming support** (SSE) allows simulated chunked responses for large inference tasks.

Worker Flow:

1. API receives inference request → validated by API key middleware

2. Rate limit and telemetry updated

3. Request added to `inference` queue

4. Worker picks up job → simulates GPU processing → stores output

5. API returns result to client (either via normal JSON or SSE streaming)

---

4. Trade-offs & Assumptions

- **Models**: Stored in-memory for simplicity; no real AI model inference.

- **GPU simulation**: Random delay mimics compute time.

- **API keys & metrics**: Stored in Redis and in-memory; persistent storage not implemented.

- **Streaming**: Simple SSE implementation; no chunk-level backpressure.

- **Single Redis instance**: Limits throughput; in production, Redis Cluster recommended.

---

5. Scalability Considerations

- **Workers**: Can be scaled horizontally by adding more Node instances with BullMQ.
- **Redis**: Centralized queue and metrics store; supports sharding/replication in production.
- **Backend**: Stateless; can horizontally scale behind load balancers.
- **Frontend**: Fully static SPA; can be served from CDN or simple Node/NGINX server.

---

6. Areas for Future Improvement

1. **Persistent Storage**: Store API keys, models, and metrics in MongoDB or Postgres.
2. **Real GPU Integration**: Connect real model inference using PyTorch or TensorRT.
3. **Authentication & Users**: Add account-based isolation for multi-tenancy.
4. **Streaming Enhancements**: WebSocket streaming for real-time token generation.
5. **Monitoring & Logging**: Integrate Prometheus/Grafana for production-grade telemetry.
6. **Deployment Automation**: Docker Compose / Kubernetes for full stack deployment.

---

7. Summary

Nebula GPU Cloud demonstrates a simplified GPU cloud service platform with full-stack capabilities and distributed task handling. It provides:

- API key management with usage tracking and rate limiting
- OpenAI-style inference endpoints with streaming support
- Simulated GPU workers for asynchronous task processing
- Frontend dashboard for metrics, model deployment, and chat

This architecture ensures modularity, scalability, and a clear foundation for future production-grade enhancements.