16. Performance Optimization in Node.js

As Node.js is **single-threaded** by default, performance optimization is crucial to handle large-scale applications. Let's break it down:

---

1. Optimize Asynchronous Code

- Always use **async/await** or **Promises** instead of blocking synchronous functions.
- Example:

```
// ❌ Bad: Blocking
const data = fs.readFileSync("file.txt", "utf8");

// ☑ Good: Non-blocking
const data = await fs.promises.readFile("file.txt", "utf8");
```

---

2. Use Caching

- **In-memory cache**: Store frequently used results.
- **Redis/Memcached**: For distributed caching across multiple servers.
- Example: Cache DB query results to reduce DB load.

---

3. Load Balancing & Clustering

- Use **Node.js Cluster** or **PM2** to spawn multiple processes and utilize all CPU cores.
- Example:

```
const cluster = require("cluster");
const os = require("os");

if (cluster.isMaster) {
    os.cpus().forEach(() => cluster.fork());
} else {
    require("./server.js");
}
```

---

4. Optimize Database Queries

- Use **indexes** in SQL/NoSQL.

- Avoid **N+1 queries** → use `populate` in Mongoose or joins in SQL.
- Apply **connection pooling** for databases.

---

5. Use Streams for Large Data

- Streams process data in chunks instead of loading entire data into memory.
- Example:

```
const fs = require("fs");
const readStream = fs.createReadStream("bigfile.txt");
readStream.pipe(process.stdout);
```

---

6. Reduce Middleware Overhead

- Keep middleware minimal in Express.
- Example: Instead of global middleware, apply middleware only on required routes.

---

7. Gzip & Compression

- Use `compression` middleware in Express to reduce payload size.

```
const compression = require("compression");
app.use(compression());
```

---

8. Use HTTP/2

- Faster multiplexed connections, reduced latency.
- Example: Use `spdy` or native HTTP/2 in Node.js.

---

9. Implement Rate Limiting

- Prevent abuse & improve availability.

```
const rateLimit = require("express-rate-limit");
const limiter = rateLimit({ windowMs: 1 * 60 * 1000, max: 100 });
app.use(limiter);
```

---

10. Use Message Queues

- Offload heavy tasks to **background workers**.
- Tools: RabbitMQ, Kafka, Bull (Redis-based).

---

11. Optimize JSON Handling

- Large JSON parsing can block the event loop.
- Use streaming parsers like `JSONStream`.

---

12. Monitor & Profile Performance

- Tools: **Node Clinic**, **New Relic**, **PM2 monitoring**.
- Use `process.memoryUsage()` and `console.time()` for performance tracking.

☑ **Interview Tip:**

- Q: *How do you scale a Node.js application?*
    - **Answer:** Use clustering, load balancing with Nginx, caching with Redis, optimize DB queries, and offload heavy tasks to workers/message queues.

Would you like me to next cover **17. Microservices & Message Queues** (with RabbitMQ, Kafka, Redis Pub/Sub) or jump directly to **18. GraphQL with Node.js**?