Backend Mastery – Node.js, Express, DBMS, Cloud, Redis, Kafka, Production APIs

> **Complete guide for senior backend developers: all concepts, theory, practical usage, complex production-ready code, best practices, and integrations.**

1️⃣ THINGS TO KNOW & LEARN SEQUENTIALLY

1. **Node.js Core Concepts**
2. Event loop, async patterns, streams, modules
3. **Express.js Basics**
4. Routing, middleware, error handling
5. **Database Concepts**
6. SQL / PostgreSQL / MongoDB
7. Advanced queries, joins, aggregation, indexing, transactions
8. **Authentication & Authorization**
9. JWT, OAuth, role-based access
10. **Caching**
11. Redis, cache patterns, middleware
12. **File Handling & Cloud Storage**
13. Multer, AWS S3, Cloudinary
14. **Advanced API Patterns**
15. Optimized pagination, aggregation, batch processing
16. **Messaging & Queues**
17. Kafka / RabbitMQ, event-driven architecture
18. **Cron Jobs & Scheduling**
19. node-cron, agenda, scheduling jobs
20. **Mailing & Notifications**
    - Nodemailer, transactional emails, OTP systems
21. **Webhook Handling**
    - Third-party webhooks, signature verification
22. **Logging & Monitoring**
    - Winston, Pino, Sentry, Prometheus/Grafana
23. **Production Best Practices**
    - Clustering, PM2, security, rate limiting, error handling, configuration management

---

2️⃣ DETAILED TOPICS WITH THEORY, USAGE & PRODUCTION CODE

2.1 Node.js Core Concepts

Theory: - Non-blocking, single-threaded runtime - Event-driven I/O for scalability

Usage & Code:

```
const fs = require('fs');
fs.readFile('data.txt', 'utf-8', (err, data) => { if(err) throw err;
console.log(data); });
```

Why: Efficient handling of concurrent requests without blocking the main thread.

2.2 Express.js

Theory: - Minimalist web framework for Node.js - Middleware-based request handling

Code:

```
const express = require('express');
const app = express();
app.use(express.json());
app.get('/status', (req,res)=>res.json({status:'OK'}));
app.listen(3000);
```

Why: Structure backend APIs in a maintainable, modular way.

2.3 Database Integration

- **SQL/PostgreSQL**: relational, complex joins, transactions, window functions
- **MongoDB**: NoSQL, aggregation pipelines, flexible schema, transactions

Complex Query Example (PostgreSQL):

```
SELECT u.id, u.name, SUM(o.total) AS total_spent
FROM users u
JOIN orders o ON u.id = o.user_id
WHERE o.status='PAID'
GROUP BY u.id;
```

MongoDB Equivalent:

```
User.aggregate([
  { $lookup: { from:'orders', localField:'_id', foreignField:'userId',
as:'orders' } },
  { $unwind:'$orders' },
  { $match:{ 'orders.status':'PAID' } },
  { $group:{ _id:'$_id', totalSpent:{ $sum:'$orders.total' } } }
]);
```

2.4 Authentication & Authorization

- **JWT Example:**

```
const token = jwt.sign({ userId: user._id }, 'secret', {
expiresIn:'1h' });
```

- **Role-based access middleware:**

```
const auth = (roles) => (req,res,next)=>{ if(!
roles.includes(req.user.role)) return
res.status(403).send('Forbidden'); next(); };
```

2.5 Caching with Redis

```
const cacheMiddleware = async (req,res,next) => {
  const data = await redisClient.get(req.originalUrl);
  if(data) return res.json(JSON.parse(data));
  next();
};
```

Why: Reduces DB load, speeds up APIs.

2.6 File Handling & Cloud Storage

- **Multer for uploads**
- **AWS S3 Upload**

```
await s3.upload({ Bucket:'bucket', Key:file.originalname,
Body:file.buffer }).promise();
```

- **Cloudinary Upload**

```
const result = await cloudinary.uploader.upload(file.path);
```

2.7 Advanced API Patterns

- Cursor-based pagination
- Aggregation pipelines
- Batch processing
- Async error handling wrapper

```
const asyncHandler = fn =>
(req,res,next)=>Promise.resolve(fn(req,res,next)).catch(next);
```

2.8 Messaging & Queues

- **Kafka Producer:** send OTP/email events
- **Kafka Consumer:** process events asynchronously

```
producer.send({ topic:'otp', messages:[{ value: JSON.stringify({
otp,userId }) }] });
```

2.9 Cron Jobs & Scheduling

```
cron.schedule('0 0 * * *', async()=>{ await cleanupOldLogs(); });
```

2.10 Mailing & OTP System

- Nodemailer for transactional emails
- OTP generated, sent via Kafka, stored in Redis with TTL

```
await transporter.sendMail({ to:user.email, subject:'OTP', html:`<p>$
{otp}</p>` });
```

2.11 Webhook Handling

```
app.post('/webhook/payment', express.json(), async(req,res)=>{
  await PaymentService.handle(req.body);
  res.sendStatus(200);
});
```

2.12 Logging & Monitoring

- Winston/Pino for structured logging
- Sentry/Prometheus for error monitoring & metrics

2.13 Production Best Practices

- Environment variables management
- Clustering / PM2
- Compression & security middleware
- Rate limiting
- Health checks
- Exception handling & graceful shutdown

📌**This documentation sequentially covers everything a senior backend engineer must know: theory, concepts, why we use it, how to use it, and complex production-ready Node.js/Express code.**