Node.js – Expanded Notes

---

1. Overview

Node.js is:

- A **JavaScript runtime** built on **Chrome's V8 engine**.
- Used to **execute JavaScript on the server-side**.
- Designed to build **fast, scalable network applications**.
- Maintains a **non-blocking, event-driven architecture**, which is ideal for I/O-heavy operations like APIs and real-time services.

Why Node.js?

- Unified language (JavaScript) across front-end and back-end.
- Lightweight and fast.
- Large ecosystem via **npm**.
- Ideal for **real-time apps**, **microservices**, and **REST APIs**.

---

2. Architecture

Event-Driven, Non-blocking I/O

- **Single-threaded Event Loop**: Unlike traditional multi-threaded servers (like Java or PHP), Node.js runs on a single thread and uses **asynchronous callbacks** to manage concurrency.

Event Loop Phases

- **Timers** → **I/O Callbacks** → **Idle/Prepare** → **Poll** → **Check** → **Close Callbacks**
- **libuv**: The library Node.js uses to abstract asynchronous I/O operations.

Worker Threads

- Introduced to support **CPU-intensive** tasks (v10.5+).
- Useful for parallelizing expensive operations (like image processing, ML).

---

3. npm (Node Package Manager)

What is npm?

- The **default package manager** for Node.js.
- Hosts 2M+ open-source packages.

Common Commands:

```
npm init            # Initialize project
npm install <pkg>    # Install dependency
npm install -g <pkg>  # Install globally
npm run <script>     # Run script in package.json
```

Key Files:

- package.json: Lists project metadata and dependencies.
- package-lock.json: Locks dependency versions for reproducible installs.

---

4. Modules & File Structure

Module Systems:

- **CommonJS** (require) — default in Node.js.
- **ES Modules (ESM)** (import) — supported using "type": "module" in package.json.

Example (CommonJS):

```
// calc.js
function add(a, b) {
  return a + b;
}
module.exports = add;


// app.js
const add = require('./calc');
console.log(add(5, 3));
```

Built-in Core Modules:

Module	Description
fs	File system operations
http	Create HTTP server
path	File paths handling
os	System-level info
events	Event emitter functionality
crypto	Hashing, encryption
stream	Stream-based I/O

---

5. Creating Servers with Node.js

Using http module:

```
const http = require('http');


const server = http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World');
});


server.listen(3000);
```

Using Express.js (more commonly in real-world apps):

```
const express = require('express');
const app = express();


app.get('/', (req, res) => res.send('Hello Express!'));
app.listen(3000);
```

---

6. File System with fs

```
const fs = require('fs');


// Async read
fs.readFile('text.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log(data);
});


// Sync write
fs.writeFileSync('output.txt', 'Hello File');
```

---

7. Asynchronous Programming Patterns

Callback

```
fs.readFile('file.txt', (err, data) => {
  if (err) return console.error(err);
  console.log(data);
});
```

Promises

```
const fs = require('fs/promises');
fs.readFile('file.txt', 'utf8')
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

Async/Await

```
const readFile = async () => {
  try {
    const data = await fs.readFile('file.txt', 'utf8');
    console.log(data);
  } catch (err) {
```

```
  console.error(err);
 }
};
```

---

8. Environment Variables

- Store sensitive data like API keys and DB credentials.
- Use .env files with dotenv package.

```
# .env
PORT=3000
require('dotenv').config();
console.log(process.env.PORT);
```

---

9. Middleware in Express

- Middleware functions access req, res, and next().
- Use them for logging, authentication, request parsing, etc.

```
app.use((req, res, next) => {
 console.log(`${req.method} - ${req.url}`);
 next();
});
```

---

10. Testing in Node.js

Tool	**Use**
Jest	All-in-one test runner
Mocha	Flexible test framework
Chai	Assertion library
Supertest	HTTP assertions for APIs

---

11. Tools & Best Practices

Developer Tools:

- nodemon: Auto-restarts server on file changes.
- eslint: Linting and code quality.
- Prettier: Code formatting.
- pm2: Production-grade process manager.

Best Practices:

- Use **async/await** over callbacks.
- Handle **all errors** properly.
- Keep code **modular** and **layered** (routes, controllers, services).
- Use **middleware** for cross-cutting concerns.
- Store **secrets in env variables**, not in code.
- Log efficiently using tools like **Winston** or **Morgan**.

---

12. Common Use Cases

Use Case	**Description**
REST APIs	JSON-based APIs for web/mobile apps
Real-time apps	Chat, notifications via WebSockets (e.g., socket.io)
Command-line tools	Tools like npm itself
Microservices	Independent services communicating via HTTP/Queues
Serverless functions	Deploy via AWS Lambda, Google Cloud Functions, etc.