Node.js - Practice Sheet

10 Q&A on event loop, streams, clustering, and async patterns

Q1. What are the phases of the Node.js event loop?

A: Timers (setTimeout/setInterval callbacks), Pending callbacks (I/O errors), Idle/Prepare (internal), Poll (I/O events, blocking if nothing else pending), Check (setImmediate callbacks), Close callbacks. Each phase has a queue that drains before moving to the next.

Q2. What is the difference between process.nextTick(), setImmediate(), and setTimeout(fn,0)?

A: process.nextTick() runs before any I/O callbacks, at the end of the current operation highest priority. setImmediate() runs in the Check phase after I/O callbacks. setTimeout(fn,0) runs in the Timers phase, which may be after I/O depending on timing.

Q3. How does Node.js clustering work?

A: cluster.fork() creates child worker processes that all share one TCP server port. The master distributes incoming connections (round-robin on most OS). Each worker has its own V8 instance and event loop. cluster.isMaster / cluster.isWorker distinguish roles.

Q4. What is backpressure in Node.js streams and why does it matter?

A: Backpressure occurs when a Writable stream cannot consume data as fast as a Readable produces it. If ignored, memory grows unboundedly. pipe() handles backpressure automatically by pausing the Readable when the Writable's internal buffer is full.

Q5. What is the difference between CommonJS require() and ES module import?

A: CommonJS uses require() (synchronous, dynamic, returns a copy). ES modules use import/export (static, asynchronous, live bindings). Node.js supports both: .cjs for CommonJS, .mjs or 'type':'module' in package.json for ESM. They cannot be mixed without interop care.

Q6. How do you handle errors in async Node.js code?

A: Callback style: check the first err argument. Promises: .catch() or try/catch in async functions. Unhandled rejections can crash the process in newer Node versions. Use process.on('uncaughtException') and 'unhandledRejection' as last-resort safety nets.

Q7. What is the Node.js Buffer class and when is it needed?

A: Buffer represents fixed-size binary data outside the V8 heap (no GC pressure). It is used when working with raw TCP streams, file system data, or binary protocols where encoding is not UTF-8. Buffer.from(data, 'base64') and buf.toString('hex') convert formats.

Q8. When should you use Node.js worker_threads vs cluster?

A: cluster forks separate processes sharing a port ideal for scaling I/O-bound HTTP servers across CPUs. worker_threads run in the same process with shared ArrayBuffer memory ideal for CPU-intensive tasks (image processing, cryptography) without IPC serialization overhead.

Q9. What operations use the libuv threadpool in Node.js?

A: File system operations (fs.*), DNS lookups (dns.lookup), and crypto operations (bcrypt, randomBytes in older versions) use the 4-thread libuv pool by default. Increase with UV_THREADPOOL_SIZE=8 if many blocking ops run concurrently.

Q10. How do you profile a Node.js application for performance?

A: Use node --prof app.js to generate a V8 isolate log, then node --prof-process to summarize hot functions. Chrome DevTools (node --inspect) provides a CPU flame graph. clinic.js doctor and flame give guided profiling for production diagnostics.