Node.js Core & Internals

1. What is the Node.js event loop? ➝ An asynchronous, non-blocking loop that handles all callbacks and async operations.

2. Explain event loop phases. ➝ Timers → Pending callbacks → Idle/prepare → Poll → Check → Close callbacks.

3. Difference between process.nextTick() and setImmediate()? ➝ nextTick runs before the event loop continues; setImmediate runs in the check phase.

4. What are microtasks and macrotasks? ➝ Microtasks (promises, nextTick) run immediately after current task; macrotasks (setTimeout, setImmediate) run in next loop phase.

5. What are Worker Threads in Node.js? ➝ A way to run CPU-intensive JavaScript in parallel threads.

6. Difference between cluster and worker threads? ➝ Cluster creates multiple Node.js processes; worker threads share memory inside one process.

7. How does Node.js handle I/O? ➝ Through libuv's event loop and thread pool (for filesystem, DNS, crypto).

8. What is libuv? ➝ A C library providing event-driven async I/O for Node.js.

9. How does garbage collection work in Node.js? ➝ V8 uses generational GC with a young and old heap.

10. What is the difference between CommonJS and ES Modules? ➝ CommonJS is synchronous (require), ESM is asynchronous (import/export).

Asynchronous Patterns

11. Callback hell problem? → Nesting callbacks makes code unreadable; solved with Promises/async-await.

12. Difference between Promise.all and Promise.allSettled? → all fails on first rejection; allSettled waits for all results.

13. What is Promise.race? → Resolves/rejects with the first settled promise.

14. What is async/await syntactic sugar for? → For promises, allowing async code to look synchronous.

15. How to handle errors in async/await? → Use try/catch or .catch().

Streams & Buffers

16. What are Node.js streams? → Objects that let you process data piece by piece instead of loading everything into memory.

17. Types of streams? → Readable, Writable, Duplex, Transform.

18. What is backpressure? → When data production is faster than consumption, streams apply flow control.

19. Difference between fs.readFile and fs.createReadStream? → readFile loads entire file into memory; createReadStream processes chunks.

20. Use case of buffers? → To handle binary data (files, TCP, streams).

API Design & Architecture

21. Best practices for REST API design? → Use nouns, versioning, pagination, filtering, proper status codes, error handling.

22. When to use GraphQL vs REST? → GraphQL when clients need flexible queries; REST for simpler, well-defined APIs.

23. What is an API Gateway? → A single entry point for routing, authentication, rate limiting, logging.

24. What are microservices? → Small, independent services communicating via APIs/events.

25. What is service discovery? → A way for microservices to find each other dynamically (e.g., Consul, Eureka).

Scaling & Performance

26. How do you scale Node.js apps? → Clustering, load balancers, Docker/K8s, caching, CDNs.

27. Difference between horizontal and vertical scaling? → Horizontal = more machines; vertical = more resources on one machine.

28. What is PM2? → A process manager for Node.js with clustering, monitoring, restarts.

29. How do you handle millions of requests per day in Node.js? → Load balancing, caching, async non-blocking code, DB optimization.

30. How do you optimize event loop performance? → Avoid blocking operations, use streams, worker threads for CPU-heavy tasks.

Caching & Queues

31. Why use caching? → Reduce DB load, improve response time.
32. Types of caching? → In-memory, Redis/Memcached, CDN, application-level.
33. What is Redis used for in Node.js? → Session store, caching, pub/sub, rate limiting.
34. What are message queues? → Systems like Kafka, RabbitMQ, SQS to decouple and scale async tasks.
35. What is event-driven architecture? → Services communicate by emitting and listening to events.

Databases

36. SQL vs NoSQL differences? → SQL = structured, relational, ACID; NoSQL = flexible schema, scale, BASE.

37. How does connection pooling work? → Reuses DB connections instead of creating new ones per request.

38. When would you use MongoDB over PostgreSQL? → For flexible schema, document-based storage, rapid prototyping.

39. What is an index in databases? → A data structure that speeds up queries but slows writes.

40. What are database transactions? → Group of operations that follow ACID (atomic, consistent, isolated, durable).

Security

41. What is OWASP Top 10? → List of common security risks (XSS, CSRF, SQL Injection, etc.).
42. How do you secure Node.js apps? → Helmet, CORS, rate limiting, sanitizing input.
43. What is CSRF? → Cross-site request forgery; attacker tricks user into sending requests.
44. What is XSS? → Cross-site scripting; injecting malicious scripts into web apps.
45. How do JWTs work? → Encoded JSON with signature, used for stateless authentication.
46. Difference between OAuth2 and JWT? → OAuth2 = framework for authorization; JWT = token format used in auth.
47. What is token refresh strategy? → Short-lived access tokens + long-lived refresh tokens.
48. How do you prevent brute-force login attempts? → Rate limiting, account lockouts, captcha.
49. How do you manage API keys securely? → Vault services (AWS Secrets Manager, HashiCorp Vault), env vars, rotation.
50. Why is eval dangerous in Node.js? → It executes arbitrary code, prone to injection attacks.

Testing

51. Difference between unit, integration, and E2E tests? → Unit = single function; Integration = multiple modules; E2E = full flow.
52. What testing frameworks do you use in Node.js? → Jest, Mocha, Chai, Supertest.
53. What is TDD? → Test-driven development: write tests before code.
54. What is mocking in tests? → Simulating dependencies (DB, API calls).
55. What is code coverage? → Percentage of code executed during tests.

DevOps & Deployment

56. How do you deploy Node.js apps? → VMs, Docker containers, Kubernetes, Serverless.
57. What is zero-downtime deployment? → Using blue-green or canary deployments.
58. What is CI/CD? → Continuous integration and deployment pipelines.
59. What is a reverse proxy? → Nginx/HAProxy for load balancing, caching, SSL termination.
60. How do you monitor Node.js apps? → ELK stack, Prometheus + Grafana, Datadog, New Relic.

Monitoring & Debugging

61. How to debug memory leaks in Node.js? → Heap snapshots, process.memoryUsage(), Chrome DevTools, clinic.js.

62. How to profile CPU usage? → --inspect, flamegraphs, 0x, clinic.

63. What is structured logging? → JSON logs with fields (timestamp, level, requestId).

64. Difference between logs, metrics, and tracing? → Logs = events; Metrics = aggregated data; Tracing = request path.

65. What is OpenTelemetry? → Standard for distributed tracing and observability.

System Design

66. How would you design a chat app in Node.js? → WebSockets, Redis pub/sub, MongoDB for persistence, scaling with load balancer.

67. How to design an e-commerce system? → Services: catalog, cart, order, payment; DB sharding; caching; eventual consistency.

68. How do you design rate limiting? → Fixed window, sliding window, token bucket using Redis.

69. How do you scale WebSocket servers? → Redis pub/sub, sticky sessions, horizontal scaling with load balancer.

70. How to design a video streaming service? → CDN for delivery, chunked uploads, adaptive bitrate streaming.

Advanced Topics

71. What is backpressure in event loop? → Overload when async operations queue faster than handled.

72. How to implement graceful shutdown in Node.js? → Handle SIGTERM/SIGINT, close DB connections, finish requests.

73. What is a circuit breaker pattern? → Stops calls to failing service to avoid overload.

74. What is a saga pattern? → Distributed transaction handling using events + compensation.

75. How do you prevent cascading failures in microservices? → Circuit breakers, retries, timeouts, bulkheads.

76. What is API throttling? → Limiting API usage to prevent abuse.

77. Difference between rate limiting and throttling? → Rate limiting = max requests/time; throttling = slowing requests.

78. What is Canary Deployment? → Gradually rolling out new features to subset of users.

79. What is Blue-Green Deployment? → Switching traffic between two identical environments.

80. What is Backoff Retry strategy? → Retrying failed requests with increasing delays.

81. What are feature flags? → Toggle features at runtime without redeploying.

82. What is CAP theorem? → In distributed systems: Consistency, Availability, Partition tolerance (pick 2).

83. Difference between Monolith and Microservices? → Monolith = single codebase; Microservices = independent services.

84. What is a Bulkhead pattern? → Isolate failures to one component to protect system.

85. How does sharding work in databases? → Splitting data across multiple DB instances.

86. Difference between Horizontal vs Vertical Partitioning? → Horizontal = splitting rows; Vertical = splitting columns.

87. What is Idempotency in APIs? → Same request can be made multiple times without changing result.

88. How do you ensure backward compatibility in APIs? → Versioning, deprecation strategy.

89. What is gRPC and why use it? → A high-performance RPC framework using HTTP/2 and protobuf.

90. Difference between REST and gRPC? → REST = text (JSON), flexible; gRPC = binary (protobuf), faster.

91. How does Node.js handle SSL/TLS? → Using built-in crypto & https modules.

92. What is a Reverse Proxy in Node.js context? → Middleware/NGINX that forwards client requests to backend services.

93. What is sticky session in load balancing? → Ensures same client always connects to same server instance.

94. What is eventual consistency? → Data may be stale temporarily but eventually becomes consistent.

95. Difference between Strong vs Eventual consistency? → Strong = immediate consistency; Eventual = eventual convergence.

96. What is CQRS? → Command Query Responsibility Segregation: separate read/write models.

97. What is Event Sourcing? → Store state as sequence of events instead of final state.

98. What are Webhooks? → HTTP callbacks triggered by events.

99. Difference between WebSocket and SSE? → WebSocket = full-duplex; SSE = server push only.

100. How do you secure WebSocket connections? → TLS (wss://), authentication, rate limiting, token validation.