Embedded C/C++ - Practice Sheet

10 conceptual Q&A IoT & Edge

Q1. What is Embedded C/C++ and what problem in IoT & Edge does it solve?

A: Embedded C/C++ is a tool or platform that automates or simplifies a specific concern in the IoT & Edge domain, reducing manual effort and operational complexity for engineering teams.

Q2. What are the core components or concepts in Embedded C/C++?

A: Embedded C/C++ has key building blocks that work together: a configuration layer defines intent, a runtime layer enforces it, and an API or CLI layer allows operators to manage and observe the system.

Q3. How does Embedded C/C++ compare to its main alternatives?

A: Embedded C/C++ trades off simplicity against feature richness differently than its alternatives. Choose Embedded C/C++ when its specific strengths performance, ecosystem, or ease of use align with your team's primary constraints.

Q4. How do you get started with Embedded C/C++ for a new project?

A: Install Embedded C/C++ via its package manager or binary release. Follow the quickstart to initialize a project, configure the minimal required settings, and run the default command to verify the setup works end-to-end.

Q5. What configuration options most affect Embedded C/C++'s behavior in production?

A: Production-critical settings typically include concurrency limits, timeout values, retry policies, resource limits, and logging levels. Review the official production hardening guide before deploying.

Q6. How does Embedded C/C++ handle reliability and high availability?

A: Embedded C/C++ provides HA through leader election, replication, or stateless horizontal scaling. Deploy at least two instances across availability zones and configure health checks for automatic failover.

Q7. What observability does Embedded C/C++ provide out of the box?

A: Embedded C/C++ exposes metrics (Prometheus endpoint or built-in dashboard), structured logs, and optionally distributed traces. Define SLOs on the key metrics and alert before users are affected.

Q8. What are common performance bottlenecks in Embedded C/C++?

A: Performance bottlenecks typically stem from misconfigured connection pools, large payload sizes, insufficient worker threads, or missing caches. Profile with built-in diagnostics before optimizing.

Q9. What security best practices apply when running Embedded C/C++?

A: Run Embedded C/C++ with the principle of least privilege, enable TLS for all communication, use a secrets manager for credentials, and keep the software patched to the latest stable release.

Q10. What mistakes do teams commonly make when adopting Embedded C/C++?

A: Common pitfalls include skipping the documentation and learning the API by trial and error, using default configurations in production, lacking a runbook for operational issues, and not testing failover scenarios.