Distributed Transactions - Practice Sheet

10 conceptual Q&A Distributed Systems

Q1. What is Distributed Transactions and what problem in Distributed Systems does it solve?

A: Distributed Transactions is a tool or platform that automates or simplifies a specific concern in the Distributed Systems domain, reducing manual effort and operational complexity for engineering teams.

Q2. What are the core components or concepts in Distributed Transactions?

A: Distributed Transactions 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 Distributed Transactions compare to its main alternatives?

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

Q4. How do you get started with Distributed Transactions for a new project?

A: Install Distributed Transactions 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 Distributed Transactions'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 Distributed Transactions handle reliability and high availability?

A: Distributed Transactions 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 Distributed Transactions provide out of the box?

A: Distributed Transactions 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 Distributed Transactions?

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 Distributed Transactions?

A: Run Distributed Transactions 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 Distributed Transactions?

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.