Interview Questions – Backup & Recovery Strategies

✅ 1. What is the difference between full, incremental, and differential backups? When would you use each?

- **Full Backup**:
    - Backs up **all data**, regardless of changes.
    - Pros: Easy to restore.
    - Cons: Time-consuming and requires high storage.
    - **Use case**: Weekly full backups as a foundation.
- **Incremental Backup**:
    - Backs up only the **data that changed since the last backup** (incremental or full).
    - Pros: Fast and storage-efficient.
    - Cons: Recovery requires all incremental backups.
    - **Use case**: Daily backups after a full weekly backup.
- **Differential Backup**:
    - Backs up **changes since the last full backup**.
    - Pros: Faster restore than incremental.
    - Cons: Larger than incremental as the week progresses.
    - **Use case**: Mid-week restore-friendly backups.

✅ 2. How do you define and balance RTO and RPO in a large-scale distributed system?

- **RTO (Recovery Time Objective)**: Max acceptable time to restore service after failure.
- **RPO (Recovery Point Objective)**: Max acceptable data loss measured in time (e.g., last 15 mins).
- **Balancing**:
    - Low RTO/RPO → Requires hot backups, replication, and higher cost.
    - Analyze SLA commitments and criticality of components.
    - Use **tiered strategy**:
        - Mission-critical: low RTO/RPO (e.g., failover DB, real-time replication).
        - Less-critical: higher RTO/RPO (e.g., batch systems, cold backups).

✅ 3. Explain cold, warm, and hot recovery strategies with examples.

- **Cold Recovery**:
    - No pre-configured resources; systems must be rebuilt from backups.
    - 🧊 *Example*: Backup stored on tape or cold cloud storage.
    - Slowest, cheapest.
- **Warm Recovery**:
    - Some components (e.g., data, configs) pre-provisioned, but app not running.
    - 🔥 *Example*: Standby server with recent backups, manual DB restore.
    - Moderate speed & cost.
- **Hot Recovery**:
    - Fully functional redundant system with real-time syncing.
    - ⚡ *Example*: Active-active databases, failover-ready load-balanced clusters.
    - Fastest, most expensive.

✅ 4. How would you implement a backup strategy for a microservices-based application hosted in the cloud?

1. **Identify critical services and data stores** (DBs, object storage, configs).
2. Use **cloud-native backup tools** (e.g., AWS Backup, GCP snapshots).
3. Apply **different backup frequencies** per service:
    - Full weekly, daily incrementals.
    - Real-time replication for critical DBs.
4. Automate with IaC or CI/CD pipelines (e.g., Terraform, GitHub Actions).
5. Encrypt backups and store in **multi-region S3/Blob buckets**.
6. Regularly **test restoration** (chaos engineering / DR drills).

✅ 5. What trade-offs do you consider when designing a backup and recovery system for a high-availability service?

- **Cost vs. recovery speed**: Hot backups increase cost.
- **Complexity vs. maintainability**: Incremental backups are efficient but harder to restore.
- **Storage vs. retention**: Long retention increases storage needs.
- **Compliance vs. agility**: Regulatory backups might need immutability and longer archives.
- **RTO/RPO**: Must match business tolerance.

👉 Use **multi-tiered strategy** to align criticality with the backup effort.

✅ 6. How does cloud storage simplify or complicate backup and recovery strategies?

Simplifies:

- Elastic, durable storage (e.g., S3, Azure Blob).
- Built-in snapshotting (e.g., EBS, RDS).
- Lifecycle management (automatic tiering, archival).
- Geo-redundancy support.

Complicates:

- Vendor lock-in risks.
- Costs can spiral with high frequency or long retention.
- Need for **access management & encryption**.
- Cross-region data compliance challenges.

✅ 7. What are some best practices for backup automation and testing in production systems?

- **Automate** backups with scheduled jobs or cloud-native tools.
- Use **infrastructure as code** to provision backup policies.
- **Test restore procedures** regularly (runbook + chaos testing).
- Monitor backup success/failures via alerts.
- Encrypt backups and verify data integrity.
- Follow **3-2-1 Rule**:
    - 3 copies
    - 2 different media
    - 1 offsite (e.g., cloud or DR site)

✅ 8. How would you handle backup for a database with terabytes of data and minimal allowed downtime?

- Use **point-in-time recovery (PITR)** if supported (e.g., MySQL binlogs, PostgreSQL WAL).
- Leverage **incremental or log-based backup** instead of full backups daily.
- Use **replication (read-replica)** to offload backups.
- Take **online snapshots** (e.g., EBS or managed RDS snapshots).
- Compress, encrypt, and store to cold + warm tiers.
- Use **parallelism** and **dedicated backup windows** to minimize performance impact.