How will you plan and implement rollback strategies for RDS ?

---

1) Principles first (what rollback must guarantee)

- **Minimal data loss** — prefer PITR or replica-based rollback to avoid losing recent writes.
- **Minimal downtime** — aim for quick DNS/app cutover or promotion of a replica.
- **Safe and tested** — every rollback path must be practiced in staging.
- **Clear owner & runbook** — who executes, who approves, who monitors.

Whenever I plan a database change or migration, I always think of rollback from the start. My goal is simple — if something goes wrong, I should be able to bring the system back to a good state quickly and safely.

First, before any schema change or deployment, I always take a **manual snapshot** of the RDS instance. This gives me a clean restore point.

I also make sure **automated backups** and **point-in-time recovery** are enabled, so I can restore the DB to any exact time if needed.

Now, if an issue happens after deployment, I choose the rollback method based on what broke:

1. **If it's just an app issue** (like a bug in the new release) — I simply roll back the app version or turn off the feature flag. No DB change is needed.
2. **If the app and DB are out of sync** (maybe the new app expects a new column) — I roll back the app and point it back to the previous DB schema. If the old schema isn't compatible, I restore from the snapshot I took before migration.
3. **If data got corrupted or deleted by mistake** — I use **Point-in-Time Restore (PITR)** to create a new RDS instance from a few minutes before the issue. Then I point the app to this new instance.
4. **If I have a read replica ready** — I can **promote the replica** to be the new primary. This is faster and usually causes less data loss.
5. **For big migrations** — I prefer **blue-green deployment**. That means I create a new RDS (blue) with the new schema, test it, and switch traffic. If something fails, rollback is easy — I just switch back to the old (green) DB.

After rollback, I always **test the key functions** — can users log in, can we fetch data, are reports working, etc. Then I monitor the logs and CloudWatch metrics for a few hours.

Finally, we do a short review: what went wrong, how can we improve the rollback plan next time, and update our runbook.
So in short —
👉 I prepare before making changes (snapshot + backups).
👉 I pick the right rollback method based on the problem.
👉 I test, monitor, and learn from every rollback."

What is the **primary goal** of a rollback plan for RDS?
A. To rebuild the RDS instance from scratch
B. To restore service quickly with minimal data loss and downtime
C. To delete corrupted databases immediately
D. To disable all automated backups
✅ **Correct Answer: B.**
💡 **Explanation:** A rollback plan ensures **fast recovery** while maintaining **data integrity** and **availability** after a failure or bad deployment.

Before making a schema change or major RDS update, what should you **always do first**?
A. Disable backups to improve performance
B. Take a manual snapshot of the RDS instance
C. Promote a replica to primary
D. Modify the DB parameter group
✅ **Correct Answer: B.**
💡 **Explanation:** Always take a **manual snapshot** before risky operations to have a **safe restore point** for rollback.

What does a **Point-In-Time Restore (PITR)** do?
A. Rolls back schema changes without data loss
B. Creates a new RDS instance from backups and logs to a specific timestamp
C. Restores the same DB in place
D. Promotes a replica automatically
✅ **Correct Answer: B.**
💡 **Explanation:** PITR restores data **to an exact second** before failure by replaying backup logs — creating a **new DB instance**.

If an application release introduces a bug but the DB schema is unchanged, what's the **fastest rollback path**?
A. PITR to before the release
B. Restore from snapshot
C. Redeploy the previous application version or disable the feature flag
D. Reboot the RDS instance
✅ **Correct Answer: C.**
💡 **Explanation:** When DB schema is still compatible, **application rollback** via **previous version** or **feature flag** is fastest — no DB rollback needed.

Which rollback method allows you to **minimize data loss** by switching to an almost-synced replica?

A. PITR restore
B. Blue/Green deployment
C. Promote a read replica
D. Manual snapshot restore

✅ **Correct Answer: C.**

💡 **Explanation: Promoting a read replica** provides a **fast failover** with minimal data loss (depending on **replication lag**).

What is the **main limitation** of restoring from a manual snapshot compared to PITR?

A. It restores slower
B. It can only restore to the exact snapshot time, not a specific second
C. It requires deletion of the existing DB first
D. It cannot be automated

✅ **Correct Answer: B.**

💡 **Explanation:** A **manual snapshot** restores DB state **exactly** as captured — PITR offers **time-based precision**.

What's the **main advantage** of a Blue/Green (clone + cutover) RDS deployment for rollback?

A. No need for backups
B. Rollback is instant — just switch traffic back to the old environment
C. Automatic data synchronization without replication lag
D. Reduces storage cost

✅ **Correct Answer: B.**

💡 **Explanation:** In **Blue/Green**, rollback is a **simple DNS or connection switch** — fast, low risk, ideal for major upgrades.
