How would you debug RDS connectivity issues or performance bottlenecks?

When I see an issue with RDS, I start with the basics — can the application even reach the database?
First, I check **connectivity**.
From an EC2 or bastion host, I'll try to connect to the DB using its endpoint and port — for example, telnet <db-endpoint> 3306.
If that fails, I look at **security groups**, **VPC routes**, and **subnet settings** to make sure traffic is allowed.
Sometimes it's just a **DNS** issue or the app is using an old cached DNS entry — restarting the app often fixes that.
If we use **IAM authentication**, I also check if the token expired. (Normally, to connect to a database (like MySQL or PostgreSQL), you use a **username** and **password** that are stored **inside the database** itself.
To solve this, **AWS allows you to use IAM authentication**. This means — instead of a fixed DB password, your app uses a **temporary token** issued by **AWS IAM** (Identity and Access Management).)
Once I confirm the app can connect, I move to **performance**.
Here I rely on **CloudWatch metrics** first — CPU usage, storage space, number of connections, and IOPS.
If CPU is very high or there's a big spike in IOPS, I open **Performance Insights** to see which queries are consuming the load.
This quickly shows if one or two queries are slowing everything down.
If queries are slow, I check the **slow query logs** or run **EXPLAIN** on the query to see if it's missing an index or doing a full table scan.
Sometimes the issue is not the query but too many open connections — so I also check the connection pool on the application side.
If it's a short-term emergency, I might take quick actions like:

- Killing a runaway query.
- Scaling the DB instance to a larger size.
- Increasing IOPS or routing some read traffic to a read replica.

After stabilizing things, I focus on **long-term fixes** — like optimizing queries, adding the right indexes, using caching (Redis), and setting up proper connection pooling.

I also tune DB parameters like buffer size or max connections and keep monitoring through CloudWatch and Performance Insights.
In short —
👉 I start with **connectivity** (network, SGs, DNS).
👉 Then I look at **metrics and logs** to find the bottleneck (CPU, IO, locks, or queries).
👉 I apply **quick fixes** to stabilize, and then **optimize** for long-term performance."

Q: What is the *first* thing you should check when an application cannot connect to an RDS instance?
A. Performance Insights for slow queries
B. Security Groups, NACLs, and port (e.g., 3306/5432)
C. Increase instance size
D. Promote a read replica
✅ **Correct: B**
Why: Connectivity failures are usually network/security issues — verify SG rules, NACLs, and the DB port before anything else.

Q: Which CloudWatch metric tells you the remaining disk capacity of your RDS instance?
A. CPUUtilization
B. DatabaseConnections
C. FreeStorageSpace
D. ReadIOPS
✅ **Correct: C**
Why: FreeStorageSpace shows available storage — alert when it's low to avoid DB stalls.

Q: Which simple command from a bastion host tests basic network reachability to a MySQL RDS endpoint?
A. ps -aux | grep mysql
B. telnet mydb.xxxx.rds.amazonaws.com 3306
C. aws rds describe-db-instances
D. SELECT 1;
✅ **Correct: B**
Why: telnet (or nc -vz) checks whether the TCP port is reachable from the network location.

Q: What does a high ReplicaLag metric indicate?
A. The primary instance is down
B. Read replica is falling behind writes — reads may be stale
C. CloudWatch is delayed
D. The DB has too many connections
✅ **Correct: B**
Why: ReplicaLag in seconds shows replication delay; high values mean stale read replicas.

Q: If CPUUtilization is low but queries are slow, what is the most likely bottleneck?
A. CPU-bound queries
B. I/O (IOPS)/disk latency or connection saturation
C. Lack of backups
D. DNS TTL issues
✅ **Correct: B**
Why: Low CPU with slow queries points to I/O bottlenecks, lock contention, or connection issues rather than compute.
I/O Bottleneck — Input/Output bottleneck
I/O means **Input/Output**, or simply — **reading and writing data** to disk.
When you run a query, the database has to:

- Read data from disk (storage)
- Or write data (like when inserting or updating rows)

If the disk is **slow**, or the DB is doing **too many reads/writes** at once, queries start waiting.
This is called an **I/O bottleneck** — the storage can't keep up.

Databases use **locks** to keep data safe when multiple users try to change it at the same time.
For example:

- Query A is updating a row in the "orders" table.
- Query B tries to update the same row at the same time.
- Query B must **wait** until Query A finishes.

If this happens too often, many queries are **waiting for locks** — this is called **lock contention**.

Q: Which RDS feature gives **query-level** visibility (top SQL, waits) to identify slow queries?
A. CloudTrail
B. Enhanced Monitoring
C. Performance Insights
D. DB Subnet Group
✅ **Correct: C**
Why: Performance Insights shows top SQL by load, wait events, and helps find problematic queries.
