Separate MariaDB connection exhaustion from a network timeout

A plugin that cannot obtain a database connection may report a timeout even when MariaDB itself is running. The delay could happen while opening a network connection, waiting for an application pool slot, negotiating TLS, or executing a query. Capture the underlying exception before changing database limits.

1. Identify where the timeout occurs

Save a redacted stack trace and the incident time. “Connect timed out” points toward the connection path; an explicit “Too many connections” response means the server was reached and refused another connection. A pool acquisition timeout can instead mean all of that application's pooled connections are occupied. None of these messages proves that a bigger pool is the right fix.

Confirm the hostname and port against the application's intended configuration without sharing its password. A successful connection from the VPS host does not prove a game container has the same route, resolver or firewall permissions.

2. Check local server state

On a database server you administer, inspect the service and listener. Adjust the service name or port for your installation. These checks do not change access rules.

```
sudo systemctl status mariadb --no-pager
sudo ss -ltnp "sport = :3306"
sudo journalctl -u mariadb --since "15 minutes ago" --no-pager
```

A listener only on loopback may be intentional. Do not change it to every public interface to make one plugin connect. Use the configured private route or ask support to review the required source and destination.

3. Read connection counters

Using an authorised database account, run these read-only statements. Local administrator socket access may be available through `sudo mariadb`; other installations need an approved login. Never put a database password directly into a command-line argument.

```
SHOW GLOBAL VARIABLES LIKE 'max_connections';
SHOW GLOBAL STATUS WHERE Variable_name IN
  ('Threads_connected','Threads_running','Max_used_connections',
   'Connection_errors_max_connections','Uptime');
```

`Threads_connected` is the current connection count, while `Max_used_connections` records a peak. The max-connections error counter helps identify rejected connections. Record two snapshots several minutes apart and note server uptime: a restart resets many counters. A peak near the configured maximum is historical evidence, not proof that the present network timeout has that cause.

4. Match database and application evidence

Compare pool size, application instance count and concurrent work. Many separate plugins or replicas can each open their own pool. Sleeping connections are not automatically faulty. Slow queries, unfinished transactions or leaked application connections can keep a small pool occupied even below the server-wide limit.

Do not kill sessions indiscriminately, restart MariaDB or raise global limits while collecting evidence. Additional connections consume resources and can intensify memory pressure. Query text and process lists can contain customer information; share only a redacted excerpt through a private ticket.

Verify the diagnosis

Correlate one failed application request with its pool metrics and the database snapshots. After the responsible administrator corrects the identified cause, verify normal application requests and

observe counters over a representative workload. Include the endpoint, UTC time and exact underlying exception when escalating.

Related: secure MariaDB exports and network timeout diagnostics.

Official references

- MariaDB connection-limit diagnostics
- MariaDB server status variables
- MariaDB max_connections system variable

Revision #1
Created 2026-11-10 10:51:41 UTC by CloudExa Hosting Admin
Updated 2026-11-10 10:51:41 UTC by CloudExa Hosting Admin