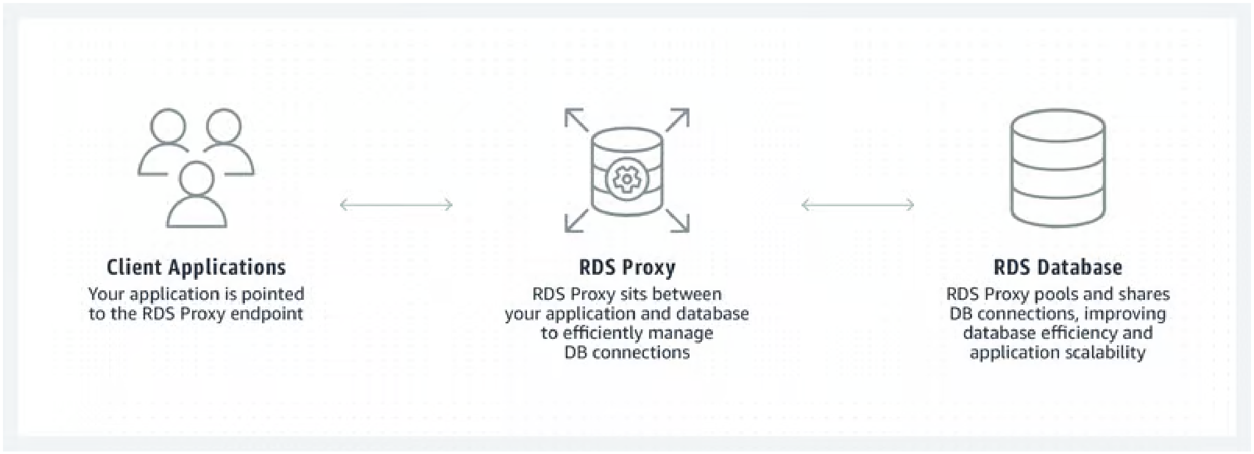
RDS Proxy and Read Replicas

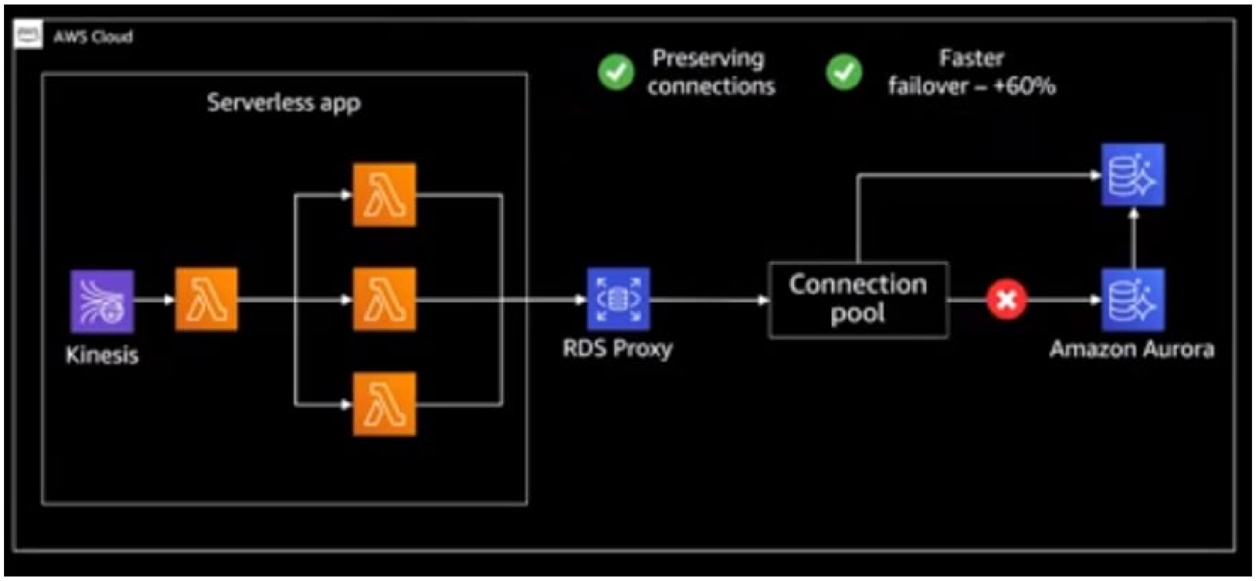
A database proxy is best understood as a middle-man that resides between an application and a database, receiving all the requests sent by the application before forwarding it to the desired database. Now, why is it exactly that we need this middle-man? What possible benefits could there be after all, to adding a extra hop between the client and the database? Well, there are a few, such as:

- **Connection Pooling**: Without a proxy, every time an application makes a request, it might need to establish a new connection to the database. This is resource-intensive, especially with high traffic. The proxy manages a pool of connections, reusing them efficiently, reducing the time and resources spent on opening and closing connections.
- **Load Balancing**: If multiple database instances exist (e.g., for replication or sharding), the proxy can distribute traffic evenly across them. This reduces the risk of overloading any single instance and ensures a smoother application performance.
- **Failover and High Availability**: In the event that a database instance goes down, the proxy can detect this and reroute traffic to a healthy instance automatically, without requiring changes on the application side. This ensures continuous availability.
- **Security**: The proxy can centralize access controls, authentication, and encryption, acting as a gatekeeper to protect the database from direct exposure. This adds an extra layer of security while streamlining how applications interact with the database.

RDS Proxy is an AWS service that provides us with a fully managed, highly available database proxy just like the one discussed above, acting as an intermediary between our application (usually also being hosted on an EC2 instance) and Amazon RDS database instances, enhancing their relationship with all the benefits mentioned above.


Client Applications
Your application is pointed to the RDS Proxy endpoint
RDS Proxy
RDS Proxy sits between your application and database to efficiently manage DB connections
RDS Database
RDS Proxy pools and shares DB connections, improving database efficiency and application scalability

The ability of the RDS Proxy to pool connections is particularly useful for application developers and cloud architects as it allows us to preserve connections and/or failover to an alternative database instance during cases where the connections between the database and the application are interrupted due to, for example, too many agents trying to connect at the same time, as shown in the architecture diagram below:



RDS Read Replicas

RDS Read Replicas are duplicates of your primary Amazon RDS database that are specifically designed to handle read-only queries. They play a key role in scaling database performance, and are particularly useful in scenarios where an application has a read-heavy workload but doesn't require comparably as many write operations such as for example, querying tools and programs.

Though they do cost extra, many organizations do not mind paying the cost due to the myriad of benefits associated with having Read Replicas such as:

1. **Offloading Read Traffic**: By creating one or more Read Replicas, organizations can direct read-only queries (like SELECT statements) to these replicas instead of burdening the primary database. This reduces the load on the primary instance, allowing it to focus more on write operations (like INSERTs, UPDATEs, or DELETEs).
2. **Horizontal Scaling**: Read Replicas enable horizontal scaling of read operations. If the number of read requests increases as traffic grows, organizations can create additional replicas to handle the increased load without affecting the performance of the primary database.
3. **High Availability for Reads**: In the event of a failure or downtime of the primary database, the Read Replicas can continue serving read requests, ensuring continuous availability for read operations.
4. **Geographic Distribution**: Read Replicas can be deployed in different regions to reduce latency for users in various geographic locations. This allows users to access data from the closest replica, improving read query response times.