# **Amazon Aurora PostgreSQL: Log-Structured Storage, Multi-AZ, and Global Database Architectures**

## **1. Introduction**

Amazon Aurora is a cloud-native relational database management system (RDBMS) designed to offer full compatibility with MySQL and PostgreSQL.1 Its fundamental objective is to merge the high performance and availability typically associated with commercial-grade databases with the operational simplicity and cost-efficiency of open-source alternatives.2 Aurora was engineered specifically for the cloud environment, aiming to overcome the inherent limitations faced by traditional database architectures in distributed settings, most notably network I/O bottlenecks.4

This document provides a technical examination of three pivotal architectural features of Amazon Aurora, with a focus on their relevance for PostgreSQL users:

1. The distinctive distributed, log-structured storage engine.
2. Multi-Availability Zone (Multi-AZ) deployments are designed for high availability within a single AWS Region.
3. Aurora Global Database, enabling cross-region disaster recovery and low-latency global read capabilities.

A comparative analysis of Multi-AZ and Global Database features will also be presented to clarify their distinct use cases and functionalities. Understanding these architectural underpinnings is essential for effectively leveraging Aurora's capabilities for demanding workloads. The design choices made in Aurora represent a significant departure from merely running PostgreSQL on cloud infrastructure; it involves a purpose-built system optimized for cloud scale, performance, and resilience.Consequently, users can anticipate different operational characteristics and performance profiles compared to standard managed PostgreSQL offerings.

## **2. Aurora's Distributed Log-Structured Storage Architecture**

The foundation of Amazon Aurora's performance and resilience lies in its unique storage architecture, which decouples compute resources (DB instances) from the storage layer, implementing them as distinct, scalable services.

### **2.1 Core Principle: Decoupling and "Log is the Database"**

Aurora fundamentally redefines the write path by adopting a "log is the database" philosophy. Unlike traditional databases that write both modified data pages and transaction logs to persistent storage, Aurora database instances transmit *only* redo log records across the network to the storage service.

Data pages are materialized from these log records within the storage layer itself. This architectural choice dramatically curtails network write traffic, often by an order of magnitude compared to traditional systems. A significant consequence of this design is the elimination of database-level checkpointing. In this process, dirty pages are flushed from the buffer cache to disk, which often causes I/O spikes and performance variability in conventional databases. By offloading log application and page materialization, Aurora minimizes write latency and improves throughput.

### **2.2 Storage Design, Replication, and Quorum System**

The Aurora storage service is a purpose-built, multi-tenant, scale-out system distributed across three distinct Availability Zones (AZs) within a single AWS Region. The database volume is logically divided into 10GB segments known as Protection Groups (PGs). Data within each PG is replicated six ways across a fleet of storage nodes, with two copies of the data residing on nodes in each of the three AZs.

This extensive replication forms the bedrock of Aurora's durability and availability. The storage volume scales automatically in 10GB increments as data grows, supporting volumes up to 128 TiB.

To manage writes and ensure data durability amidst potential failures, Aurora employs a quorum-based protocol. A write operation is acknowledged as durable only after the corresponding log record has been persisted by at least four out of the six storage nodes within the PG (a Vw​=4/6 write quorum). Reads require a

consistent version of the data to be available from at least three of the six nodes (a Vr​=3/6 read quorum). This specific 4/6 write and 3/6 read quorum configuration is deliberately chosen to provide resilience against correlated failures, specifically tolerating the failure of an entire AZ plus one additional storage node ("AZ+1" fault tolerance) without losing read availability, and tolerating the loss of any two nodes (or a full AZ) without impacting write availability.

This design acknowledges that at cloud scale, failures are often correlated (like an AZ-wide event) rather than purely independent node failures, making traditional majority quorums potentially insufficient. Furthermore, Aurora's design largely avoids costly distributed consensus protocols (like Paxos or two-phase commit) for most common operations like I/Os and commits, relying instead on the global ordering provided by monotonically increasing Log Sequence Numbers (LSNs) and local state tracking on nodes, which enhances performance, reduces variability, and lowers cost.

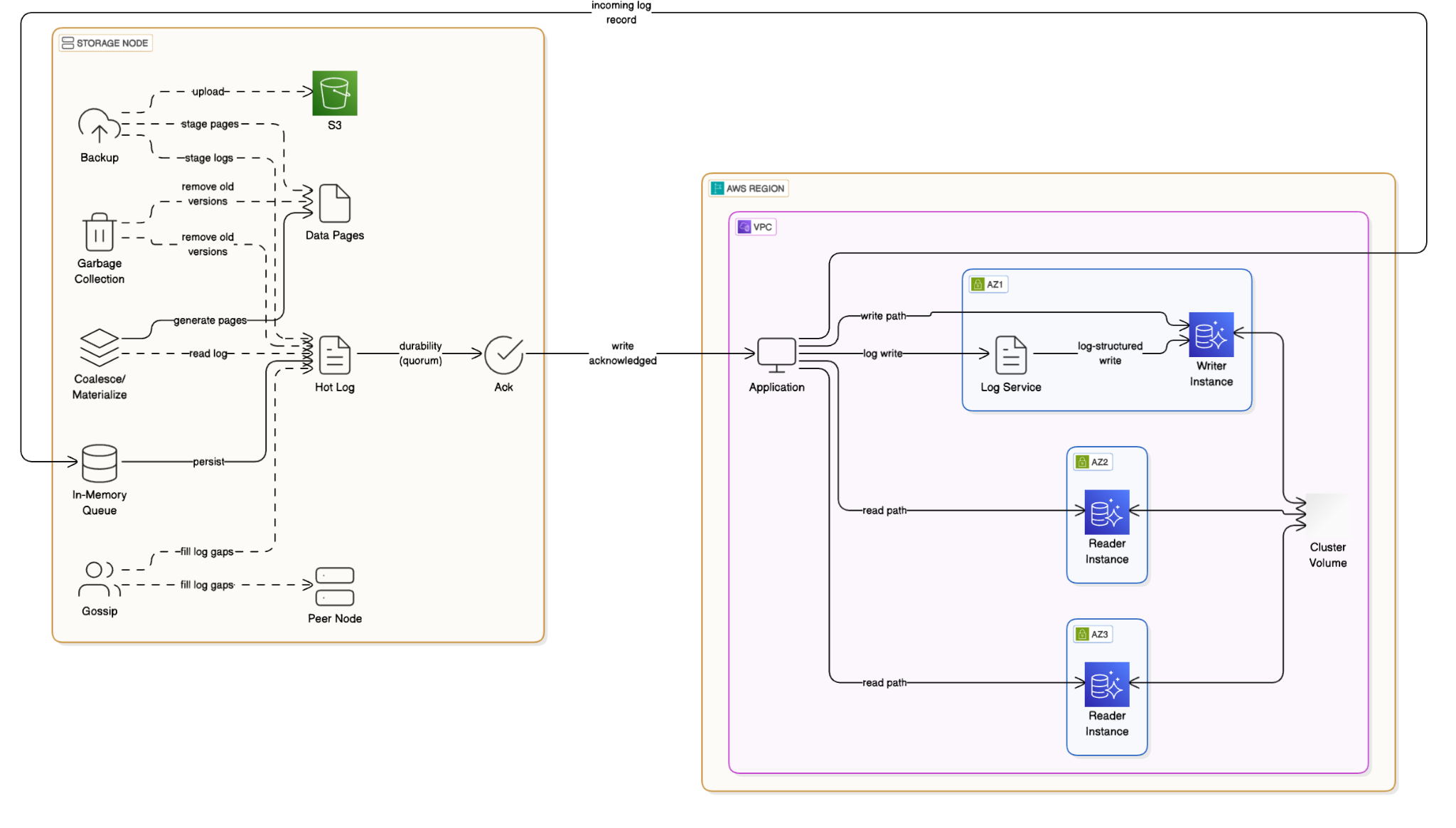
### **2.3 Write Path**

The write process in Aurora is optimized for low latency and high throughput:

1. The database instance generates redo log records for data modifications, assigning globally unique, monotonically increasing LSNs.
2. These log records—and *only* the log records—are sent in parallel to the six storage nodes responsible for the affected data segment(s). Records smaller than 4KB may be batched, while larger ones are split.
3. Each receiving storage node adds the record to an in-memory queue, performs deduplication based on LSN, and durably persists the unique log record to its local SSD-based "hot log".
4. Once four of the six nodes confirm persistence (meeting the write quorum), an acknowledgment is sent back to the database instance. Group commit mechanisms batch multiple transaction acknowledgments together, further reducing latency.

Critically, the following steps occur asynchronously in the background, *after* the write has been acknowledged to the client:

* **Page Materialization:** Storage nodes apply the persisted log records to older page versions to create new, consistent data pages.
* **Gossip Protocol:** Nodes communicate peer-to-peer to identify and retrieve any missing log records (gap filling) ensuring all nodes eventually converge.
* **Continuous Backup:** Log records and materialized pages are continuously streamed to Amazon S3 for durable backup and point-in-time recovery capabilities.
* **Garbage Collection:** Older versions of log records and data pages that are no longer needed for recovery or reads are periodically garbage collected.



*Aurora Storage Node I/O Flow:* The diagram depicts a storage node receiving incoming log records. The foreground path shows records entering an in-memory queue, being persisted to a 'Hot Log' on local storage, and triggering an acknowledgment back to the database instance once durability is achieved (implicitly, after quorum is met across nodes). The background path, operating asynchronously, shows processes like: 'Gossip' interacting with peer nodes to fill log gaps; 'Coalesce/Materialize' using the hot log to generate new 'Data Pages'; 'Backup' staging logs and pages to S3; and 'Garbage Collection' removing old versions. This visually separates the latency-critical foreground write acknowledgment from the heavier background processing.3

### **2.4 Read Path**

Read operations leverage the distributed storage system while maintaining compatibility with standard PostgreSQL page-based access:

1. When the database instance requires a data page not present in its buffer cache, it issues a read request to the storage layer, specifying the page needed and the required consistency point (LSN).2
2. Aurora intelligently routes the read request, attempting to retrieve the page from the storage node that is geographically closest or has the lowest latency, provided that node has processed all log records up to the required LSN (its local Segment Complete LSN (SCL) must be greater than or equal to the requested LSN).2
3. Typically, quorum reads are not necessary for standard data retrieval, as the database instance tracks the consistency state (SCL) of storage nodes.15
4. The selected storage node returns the appropriate materialized data page version. If the most current version hasn't been materialized yet, the node can synthesize the required page on-demand by applying relevant log records from its hot log to an older cached page version before responding.2

### **2.5 Self-Healing and Durability**

The Aurora storage layer incorporates robust self-healing capabilities. It continuously monitors the health of storage nodes and disks, automatically detecting and repairing failures.7 When a segment fails, the system automatically reconstructs the data using copies from the remaining healthy nodes, facilitated by the peer-to-peer gossip protocol.3 This repair process is rapid due to the 10GB segment size (often under a minute), minimizing the duration of reduced redundancy.3 Combined with the 6-way replication and continuous backups to Amazon S3 (designed for 99.999999999% durability), this architecture provides exceptional data durability and resilience.2 The complexity of this distributed storage system—replication, quorums, gossip, background tasks—is managed by AWS, presenting a simplified operational interface at the database level. However, understanding these underlying mechanics helps in comprehending performance characteristics and availability behaviors.7

## **3. Achieving High Availability with Aurora Multi-AZ**

While the Aurora storage layer provides intrinsic data durability and availability across AZs, ensuring high availability for the database *service* requires addressing the potential failure of the compute instances running the database engine. Aurora Multi-AZ deployments are designed for this purpose within a single AWS Region.

### **3.1 Architecture within a Region**

An Aurora cluster configured for high availability typically consists of:

* A **Primary (Writer) DB instance:** Handles all write operations (INSERT, UPDATE, DELETE) and can also serve read requests.7
* One or more **Aurora Replicas (Reader instances):** Optional (up to 15 per cluster) read-only instances that serve to scale read traffic and act as failover targets.7 These replicas asynchronously receive redo log streams from the primary.17
* The **Shared Cluster Volume:** The underlying distributed, 6-way replicated storage volume, spanning three AZs, accessed by *all* instances (primary and replicas) in the cluster.

It is crucial to understand that the storage volume is *always* Multi-AZ by design. The "Multi-AZ" deployment configuration for Aurora specifically refers to the practice of provisioning Aurora Replicas in AZs different from the primary instance to enable rapid compute failover. A single-instance Aurora cluster, while having durable Multi-AZ storage, still represents a single point of failure for compute operations.

### **3.2 Automatic Failover Mechanism**

Aurora employs an automated failover process managed by the Amazon RDS control plane:

1. **Detection:** The health of the primary DB instance is continuously monitored. Failures like underlying hardware issues, loss of instance connectivity, or even a full AZ outage impacting the primary are automatically detected.
2. **Promotion (with Replicas):** If one or more Aurora Replicas exist, RDS automatically promotes a replica to become the new primary instance. The choice of replica is determined by user-configurable promotion tiers (priority 0=highest to 15=lowest). The replica with the highest priority (lowest number) is chosen. If multiple replicas share the highest priority, the one with the largest instance size is promoted. To maximize availability, replicas should be provisioned in different AZs from the primary.
3. **Recreation (without Replicas):** If no replicas exist in the cluster, RDS attempts to create a *new* primary instance in the same AZ as the failed instance. This process takes considerably longer than promoting an existing replica.18
4. **Fast Failover Enablement:** The key to rapid failover is the shared storage volume. Since the promoted replica already has access to the current state of the data via the shared volume, no time-consuming data copying or log replay is required.17 The replica simply changes its role and begins accepting write traffic.
5. **Endpoint Management:** Applications connect via DNS endpoints. The **Cluster Endpoint** (e.g., mycluster.cluster-xxxx.us-east-1.rds.amazonaws.com) always resolves to the current primary instance. During failover, RDS automatically updates this DNS record to point to the newly promoted primary.17 The **Reader Endpoint** (e.g., mycluster.cluster-ro-xxxx.us-east-1.rds.amazonaws.com) provides load-balanced connections across all available replicas.\*\* Custom endpoints can target specific subsets of replicas.\*\*\*

### **3.3 Failover Timeline and Use Cases**

When promoting an existing replica, the failover process typically completes in **less than 60 seconds**, often **under 30 seconds**, minimizing write unavailability. If no replicas exist, the RTO increases significantly, potentially up to 10 minutes, while a new instance is created. Services like Amazon RDS Proxy can further reduce application-perceived downtime by maintaining client connections during failover and bypassing potential DNS propagation delays.

The primary use cases for Aurora Multi-AZ with replicas are:

* **Intra-Region High Availability:** Protecting applications against DB instance or AZ failure within a single region.
* **Read Scaling:** Horizontally scaling read capacity by distributing queries across replicas using the reader endpoint.
* **Reduced Maintenance Downtime:** Facilitating patching and upgrades with minimal interruption, often utilizing failover mechanisms (e.g., Zero-Downtime Patching).

Achieving the sub-minute failover RTO advertised for Aurora is therefore contingent upon provisioning and maintaining healthy replicas in different AZs; it is not an inherent property of a single-instance Aurora deployment. This necessitates factoring the cost of replica instances into HA designs.

## **4. Global Scale and Disaster Recovery with Aurora Global Database (PostgreSQL)**

For applications requiring resilience beyond a single AWS Region or needing to serve read traffic globally with low latency, Amazon Aurora Global Database provides a solution built upon the core Aurora storage architecture.

### **4.1 Cross-Region Architecture and Replication**

An Aurora Global Database consists of a single logical database that spans multiple AWS Regions. It comprises:

* **One Primary Region:** Hosts the primary Aurora cluster, which handles all write operations. This primary cluster typically includes a writer instance and potentially Multi-AZ replicas for local HA.
* **Up to Five Secondary Regions:** Each hosts a read-only replica Aurora cluster. These secondary clusters receive data replicated from the primary region and can contain up to 16 instances (1 writer placeholder, 15 readers) each to serve local read requests.

*Diagram Description (Aurora Global Database):* A representative diagram would show a box for the Primary Region (e.g., 'US East') containing an Aurora cluster (Writer + Replicas) connected to its regional Aurora Storage. Another box for a Secondary Region (e.g., 'EU West') would show a similar cluster (Reader instances only by default) connected to its regional Aurora Storage. A prominent arrow labeled "Physical Replication (Storage Layer, <1s latency)" would connect the Aurora Storage layer in the Primary Region directly to the Aurora Storage layer in the Secondary Region, indicating the dedicated, low-level replication path.30

Replication between the primary and secondary regions occurs at the storage layer, utilizing dedicated AWS infrastructure and Aurora's efficient log-shipping mechanism.26 This physical replication is asynchronous but designed for very low latency, typically **under one second**.26 Because it uses dedicated resources within the storage layer, it has minimal performance impact on the primary database cluster's active workload.26 Costs associated with Global Database include compute instances in secondary regions, cross-region data transfer fees, and charges for replicated write I/Os.9

### **4.2 Core Use Cases**

Aurora Global Database addresses two primary requirements:

* **Cross-Region Disaster Recovery (DR):** Provides business continuity in the event of a full regional outage. If the primary region becomes unavailable, a secondary region's cluster can be promoted to take over full read/write responsibilities. This architecture supports a Recovery Point Objective (RPO) typically around **1 second** (limited by the physical replication lag) and a Recovery Time Objective (RTO) of **less than 1 minute** for the promotion process itself. Different compute configurations (Provisioned, Headless, Serverless v2) can be used in secondary regions to balance RTO goals against cost.
* **Low-Latency Global Reads:** Enables applications with a global user base to serve read requests from the nearest secondary region, drastically reducing latency compared to routing all reads back to a distant primary region.

### **4.3 Failover and Switchover Processes**

Unlike the fully automatic instance failover within a Multi-AZ cluster, changing the primary region in a Global Database involves distinct processes:

* **Managed Planned Failover/Switchover:** A user-initiated, controlled procedure for planned events like DR drills or shifting the primary write location (e.g., follow-the-sun model). Aurora manages the process, ensuring the target secondary region is fully synchronized before promotion, guaranteeing zero data loss.
* **Manual Unplanned Failover:** The process invoked during an actual, unplanned outage of the primary region. This requires manual intervention or automation scripts to:
  1. Stop application writes (if possible).
  2. Identify the secondary region with the lowest replication lag (using CloudWatch metrics like AuroraGlobalDBReplicationLag ).
  3. Detach the chosen secondary cluster from the global database topology, making it an independent cluster.
  4. Promote the detached cluster to accept read/write traffic.
  5. Redirect application traffic to the new primary cluster. The RTO of less than 1 minute refers to the technical time for detachment and promotion (steps 3 & 4), but the end-to-end recovery time depends heavily on the speed of detection, decision-making, and execution of these steps.

### **4.4 Write Forwarding for PostgreSQL**

A key feature enhancing the usability of Global Database for PostgreSQL (versions 14.9+, 15.4+, etc.) is **write forwarding**. When enabled, applications connected to a read-only instance in a *secondary* region can issue DML statements (INSERT, UPDATE, DELETE). Aurora automatically intercepts these write statements and forwards them, along with necessary session context, to the writer instance in the *primary* region for execution. Once the write is committed in the primary region, the changes are replicated back to all secondary regions via the standard Global Database replication.

This capability significantly simplifies application logic for global deployments, as applications may not need to maintain separate connections or routing logic for reads versus writes based on region. However, developers must consider the implications:

* **Latency:** Forwarded writes incur cross-region network latency for the round trip to the primary region and back.
* **Consistency:** Reads performed in the secondary region immediately after a forwarded write might see stale data due to replication lag. Aurora provides session-level consistency parameters (apg\_write\_forward.consistency\_mode) allowing applications to choose between eventual consistency (default) or requesting stronger consistency, which may further increase latency.
* **Scope:** Write forwarding applies only to DML; DDL statements (e.g., CREATE TABLE) must still be executed directly against the primary cluster.

## **5. Comparison: Aurora Multi-AZ vs. Aurora Global Database**

While both Aurora Multi-AZ (using replicas) and Aurora Global Database enhance the resilience and availability of Aurora PostgreSQL clusters, they serve distinct purposes and operate at different scales. Misunderstanding their differences can lead to inadequate protection or unnecessary complexity and cost. The following table summarizes the key distinctions:

| **Feature** | **Aurora Multi-AZ (with Replicas)** | **Aurora Global Database** |
| --- | --- | --- |
| **Primary Purpose** | Intra-Region High Availability | Inter-Region Disaster Recovery & Global Reads |
| **Scope** | Single AWS Region | Multiple AWS Regions (1 Primary, up to 5 Secondaries) |
| **Architecture** | Single Cluster (Primary + Replicas), Shared Storage | Linked Primary/Secondary Clusters, Storage Replicated Across Regions |
| **Replication Scope** | Within Region (Primary to Replicas) | Across Regions (Primary Storage to Secondary Storage) |
| **Replication Type/Latency** | Asynchronous, Milliseconds | Asynchronous, Storage-based, Typically < 1 Second |
| **Failover Automation** | Automatic Instance Promotion | Managed (Planned) or Manual/Scripted (Unplanned) Region Promotion |
| **Failover Target** | Replica Instance within Primary Region | Secondary Region Cluster |
| **Typical RTO** | < 1 minute | < 1 minute (post-initiation) |
| **Typical RPO** | Near Zero / Seconds | ~1 Second (limited by replication lag) |
| **Read Scaling Focus** | Regional | Global |
| **Write Forwarding (PG)** | N/A | Yes |
| **Cost Factors** | Replica Instance Compute/License | Secondary Instance Compute/License, Cross-Region Data Transfer, Replicated Write I/Os |

**Key Differences Elaborated:**

* **Scope and Goal:** Multi-AZ focuses on surviving failures *within* a single region (instance or AZ failure), ensuring local application uptime. Global Database aims to survive the failure *of* an entire region and/or provide low-latency reads to a global audience.
* **Failover Mechanism:** Multi-AZ failover is fully automatic instance replacement. Global Database failover requires a managed or manual/scripted decision and process to promote a secondary region, reflecting the higher stakes and complexity of shifting operations across regions.31
* **RPO/RTO:** While both offer fast RTO targets (<1 min), the RPO differs significantly. Multi-AZ failover typically involves minimal data loss (seconds at most), whereas Global Database RPO is dictated by the cross-region replication lag (typically ~1 second).

It's important to note that these features are complementary. A comprehensive resilience strategy often involves deploying Aurora Global Database for cross-region DR, while *also* configuring Multi-AZ replicas within the primary (and potentially secondary) regional clusters to protect against local instance and AZ failures.26

## **6. Conclusion**

Amazon Aurora PostgreSQL leverages a fundamentally re-architected, cloud-native design, particularly its distributed, log-structured storage system, to deliver significant advantages over traditional relational databases. This storage architecture, characterized by 6-way replication across three AZs, quorum-based writes, and the offloading of redo processing, provides inherent high durability and forms the basis for Aurora's advanced features.2

Building upon this resilient storage, Aurora Multi-AZ deployments, utilizing compute replicas in different AZs, offer robust, automatic, and fast (<1 minute) failover capabilities, ensuring high availability for applications operating within a single AWS Region.

For applications demanding cross-region resilience or global read scalability, Aurora Global Database extends the core storage replication concepts across multiple AWS Regions. It provides effective disaster recovery with low RPO (~1s) and RTO (<1 min post-initiation) targets, alongside the ability to serve low-latency reads globally. The addition of write forwarding for PostgreSQL further simplifies the development of globally distributed applications.26

The combination of PostgreSQL compatibility with this suite of cloud-native features—optimized storage, automated intra-region high availability, and robust inter-region disaster recovery and global read capabilities—positions Amazon Aurora as a compelling platform for mission-critical, performance-sensitive relational database workloads operating at cloud scale. Selecting the appropriate configuration requires careful consideration of specific application requirements regarding performance, availability, geographic distribution, and cost, leveraging the distinct architectural advantages offered by Multi-AZ replicas and Global Database.

#### Works cited

1. OLTP Database, MySQL And PostgreSQL Managed Database - Amazon Aurora, accessed April 29, 2025, <https://aws.amazon.com/rds/aurora/>
2. Introducing the Aurora Storage Engine | AWS Database Blog, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/introducing-the-aurora-storage-engine/>
3. d1.awsstatic.com, accessed April 29, 2025, <https://d1.awsstatic.com/events/reinvent/2020/Amazon_Aurora_storage_demystified_DAT401.pdf>
4. Amazon Aurora: Design considerations for high throughput cloud-native relational databases, accessed April 29, 2025, <https://www.amazon.science/publications/amazon-aurora-design-considerations-for-high-throughput-cloud-native-relational-databases>
5. pages.cs.wisc.edu, accessed April 29, 2025, <https://pages.cs.wisc.edu/~yxy/cs764-f20/papers/aurora-sigmod-17.pdf>
6. Amazon Aurora development team wins the 2019 ACM SIGMOD Systems Award, accessed April 29, 2025, <https://www.allthingsdistributed.com/2019/07/aurora-sigmod-award.html>
7. ARCHIVED: Getting Started with Amazon Aurora - awsstatic.com, accessed April 29, 2025, <https://d1.awsstatic.com/whitepapers/getting-started-with-amazon-aurora.pdf>
8. Deep dive on Amazon Aurora and Amazon RDS for PostgreSQL architecture and features, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/is-amazon-rds-for-postgresql-or-amazon-aurora-postgresql-a-better-choice-for-me/>
9. Planning I/O in Amazon Aurora | AWS Database Blog, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/planning-i-o-in-amazon-aurora/>
10. Amazon Aurora: On avoiding distributed consensus for I/Os, commits, and membership changes, accessed April 29, 2025, <https://www.amazon.science/publications/amazon-aurora-on-avoiding-distributed-consensus-for-i-os-commits-and-membership-changes>
11. Amazon Aurora for Core Banking Systems | AWS for Industries, accessed April 29, 2025, <https://aws.amazon.com/blogs/industries/amazon-aurora-for-core-banking-systems/>
12. Read/Write Capability Enhancements in Amazon Aurora with Apache ShardingSphere-Proxy | AWS Partner Network (APN) Blog, accessed April 29, 2025, <https://aws.amazon.com/blogs/apn/read-write-capability-enhancements-in-amazon-aurora-with-apache-shardingsphere-proxy/>
13. Relational Database – Amazon Aurora MySQL PostgreSQL Features - AWS, accessed April 29, 2025, <https://aws.amazon.com/rds/aurora/features/>
14. pages.cs.wisc.edu, accessed April 29, 2025, <https://pages.cs.wisc.edu/~yxy/cs839-s20/papers/aurora-sigmod-18.pdf>
15. Amazon Aurora: Design Considerations + On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes - Metadata, accessed April 29, 2025, <http://muratbuffalo.blogspot.com/2022/03/amazon-aurora-design-considerations-and.html>
16. Lecture 10: Database logging, quorums, Amazon Aurora - MIT, accessed April 29, 2025, <http://nil.lcs.mit.edu/6.824/2020/notes/l-aurora.txt>
17. High availability for Amazon Aurora, accessed April 29, 2025, <https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Concepts.AuroraHighAvailability.html>
18. Is RDS Aurora Multi-AZ by default? : r/aws - Reddit, accessed April 29, 2025, <https://www.reddit.com/r/aws/comments/61u9ah/is_rds_aurora_multiaz_by_default/>
19. Why does AWS RDS Aurora have the option of "Multi-AZ Deployment" when it does replication across different zones already by default? - Stack Overflow, accessed April 29, 2025, <https://stackoverflow.com/questions/31976527/why-does-aws-rds-aurora-have-the-option-of-multi-az-deployment-when-it-does-re>
20. Amazon RDS Multi AZ Deployments | Cloud Relational Database | Amazon Web Services, accessed April 29, 2025, <https://aws.amazon.com/rds/features/multi-az/>
21. Failing over an Amazon Aurora DB cluster - Amazon Aurora, accessed April 29, 2025, <https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-failover.html>
22. Understand Amazon Aurora high availability and disaster recovery from an Oracle perspective | AWS Database Blog, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/understand-amazon-aurora-high-availability-and-disaster-recovery-from-an-oracle-perspective/>
23. Which endpoint to connect to for read/write operations using AWS Aurora PostgreSQL Database Cluster - Stack Overflow, accessed April 29, 2025, <https://stackoverflow.com/questions/62588784/which-endpoint-to-connect-to-for-read-write-operations-using-aws-aurora-postgres>
24. How do I configure AWS Aurora to separate write/read operations - DBA Stack Exchange, accessed April 29, 2025, <https://dba.stackexchange.com/questions/116361/how-do-i-configure-aws-aurora-to-separate-write-read-operations>
25. Amazon Aurora storage - AWS Documentation, accessed April 29, 2025, <https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Aurora.Overview.StorageReliability.html>
26. Amazon Aurora Global Database, accessed April 29, 2025, <https://aws.amazon.com/rds/aurora/global-database/>
27. Amazon Aurora | AWS Blog, accessed April 29, 2025, <https://aws.amazon.com/blogs/aws/category/amazon-aurora/>
28. Implementing run-time write synchronization in Amazon Aurora Global Database - AWS, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/implementing-run-time-write-synchronization-in-amazon-aurora-global-database/>
29. Diving deep into the new Amazon Aurora Global Database writer endpoint, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/diving-deep-into-the-new-amazon-aurora-global-database-writer-endpoint/>
30. Use Amazon Aurora Global Database to build resilient multi-Region applications, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/use-amazon-aurora-global-database-to-build-resilient-multi-region-applications/>
31. postgresconf.org, accessed April 29, 2025, <https://postgresconf.org/system/events/document/000/001/936/Aurora_Global_Database_Design_Patterns_for_HA_DR_final.pdf>
32. Aurora PostgreSQL Disaster Recovery solutions using Amazon Aurora Global Database - AWS, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/aurora-postgresql-disaster-recovery-solutions-using-amazon-aurora-global-database/>
33. Comparing Amazon Aurora replication solutions - AWS Prescriptive Guidance, accessed April 29, 2025, <https://docs.aws.amazon.com/prescriptive-guidance/latest/aurora-replication-options/compare-solutions.html>
34. Use Amazon Aurora Global Database to set up disaster recovery within India, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/use-amazon-aurora-global-database-to-set-up-disaster-recovery-within-india/>
35. Aurora Global Database: Multi-Region Replication - AWS, accessed April 29, 2025, <https://aws.amazon.com/awstv/watch/653d857f7fa/>
36. Cross-Region disaster recovery using Amazon Aurora Global Database for Amazon Aurora PostgreSQL - AWS, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/cross-region-disaster-recovery-using-amazon-aurora-global-database-for-amazon-aurora-postgresql/>
37. Monitor Amazon Aurora Global Database replication at scale using Amazon CloudWatch Metrics Insights - AWS, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/monitor-amazon-aurora-global-database-replication-at-scale-using-amazon-cloudwatch-metrics-insights/>
38. Amazon Aurora Global Database for PostgreSQL now supports write forwarding - AWS, accessed April 29, 2025, <https://aws.amazon.com/about-aws/whats-new/2023/11/amazon-aurora-global-database-postgresql-forwarding/>
39. Using write forwarding with Amazon Aurora Global Database for PostgreSQL, accessed April 29, 2025, <https://aws.amazon.com/blogs/database/using-write-forwarding-with-amazon-aurora-global-database-for-postgresql/>
40. Using write forwarding in an Amazon Aurora global database, accessed April 29, 2025, <https://docs.aws.amazon.com/en_us/AmazonRDS/latest/AuroraUserGuide/aurora-global-database-write-forwarding.html>