Journal of Advances in Developmental Research (IJAIDR)
E-ISSN: 0976-4844 ● Website: www.ijaidr.com ● Email: editor@ijaidr.com

Disaster Recovery Architectures and Strategies for Multi-Region Control Planes

Nikhita Kataria

Independent Researcher
Manager, Software Engineering
nikhitakataria@gmail.com

Abstract

Control plane infrastructure is a set of tier 0 critical services that are responsible for management and orchestration of internal and external applications for a software organization. Their availability, reliability and scalability is essential for enabling seamless deployments and efficient operational use.It is essential to ensure they achieve 99.99% availability and hence a robust disaster recovery (DR) strategy which can address a wide range of failure scenarios—including software bugs, hardware malfunctions, network outages, power failures, data corruption, and regional disasters. This paper outlines the design and implementation of a generic disaster recovery framework tailored for control plane services using Azure compute and storage options as examples. Key objectives include automation to eliminate manual recovery efforts, reduction in Mean Time to Recovery (MTTR), and early failure detection and response mechanisms. By standardizing tooling and enforcing high reliability principles, this strategy ensures resilience and operational continuity across distributed systems.

Keywords: Disaster recovery, control plane, replication, multi-region, Azure, MySQL, Cosmos DB.

IX. Introduction

In the context of cloud computing, the *control plane* refers to a suite of services architected to intelligently manage resources that support both offline and online applications, including customer-facing workloads. This concept often aligns with what is traditionally known as *software* or *compute infrastructure* within a software organization. While various architectural patterns exist for building control planes, certain standard configurations are commonly adopted. In the following sections, we examine these architectural models and highlight the critical importance of a robust disaster recovery (DR) strategy. An effective DR approach for control plane services should:

- Include generic, reusable tooling to facilitate recovery across all control plane components.
- Provide high availability and reliability targets, to help achieve 99.99% reachability.
- Eliminate manual intervention during recovery to ensure minimal mean time to recovery (MTTR).
- Enable rapid detection and response, adopting a "fail fast" philosophy.

X. Typical Architectures

This paper examines several control plane architectures, wherein the term *region* denotes a complete deployment of an application stack—comprising front-end, middle-tier, and storage components—

typically configured with multiple replicas to ensure availability. The architectural models under consideration include:

- **Single-Region Write and Multi-Region Read:** Applications in this category support write operations only in one region, while serving read requests from multiple geographically distributed regions.
- **Multi-Region Write and Multi-Region Read:** These applications allow both read and write operations across several regions, enabling active-active configurations to enhance availability and reduce latency.
- **Single-Region Write and Single-Region Read:** In this architecture read as well as write are done within a single region. However, similar infrastructure may be deployed in other regions for redundancy or failover purposes.

These architectures in terms of disaster recovery broadly fall into two classes of failures: *regional failures*, which means a total loss of a region, and *non-regional failures*, which cover partial loss of resources within a region. Both these categories of failures can impact application layer as well as data layer causing disruptions for the users in terms of outages or data corruption and integrity issues. To contextualize disaster recovery (DR) strategies, this paper references several Azure-provided services such as Azure Database for MySQL [1] and Cosmos DB [2] and examines containerized environments including virtual machines (VMs) and virtual machine scale sets (VMSS) [3].

XI. Multi-Region Applications Single Region Write and Multi-Region Read

Thedeployment model for these control plane applications typically involves application instances—often implemented as web services—receiving write operations in a designated *primary region*. Read operations may be served from the same primary region or from one or more *secondary regions*, which could function as disaster recovery (DR) sites or independently serve production traffic. The underlying database layer in such configurations may utilize managed services such as Azure Database for MySQL or Cosmos DB [1].Figure 1 illustrates the architectural setup referenced in the following sections.Note that the figure masks application regions being managed by the control plane application references as *app*.

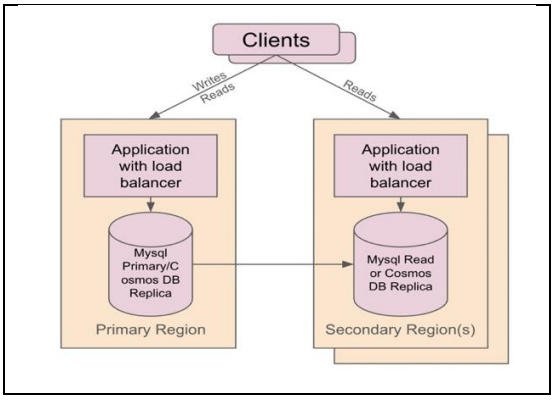

Figure 1: Typical deployment architecture with single region write and multi-region read capability.

A. *Recovery from global application outages*

The availability of an application operating in a region with no apparent infrastructure issues may still be compromised if one or more of the following failure scenarios occur concurrently.

1) ***Single application instance is affected out of the entire replica set due to hardware failure or operating system failure:*** In such cases, an automated remediation mechanism should be in place. For example, Azure's remediation service can automatically recover a failed virtual machine by relocating it to a healthy host. Additionally, it provides straightforward procedures to reboot the application environment when other healthy instances are available to continue serving traffic [4].

2) ***Single application instance is affected out of the entire replica set due to changes in application logic:*** Such failures may arise from issues introduced through application code or configuration changes. The recommended recovery approach is to either roll back to a known stable version or roll forward with a corrected deployment to remediate the software failure.

3) ***Multiple instances are affected due to an unknown infrastructure issue:***In such scenarios, the recommended response is to remove the unhealthy set of application instances from the load balancer and redirect traffic to a healthy region. It's crucial to plan for disaster recovery (DR) from the very beginning of an application's design—yet this step often gets skipped in the rush to ship new features. Major cloud providers like Microsoft Azure and AWS help here by offering load balancer health probes. These probes can automatically detect when a service is unhealthy and reroute traffic away from it. But for them to really work well, it's not enough to just check if a port is open. You also need to test whether the app is behaving correctly. The best practice is to use read-only requests that mimic real user actions, so you can spot problems without changing any data.

Figure 2 presents a summary of the mitigation strategies employed in response to partial failures, encompassing automated remediation, rollback actions, and the exclusion of unhealthy instances from the load balancer.

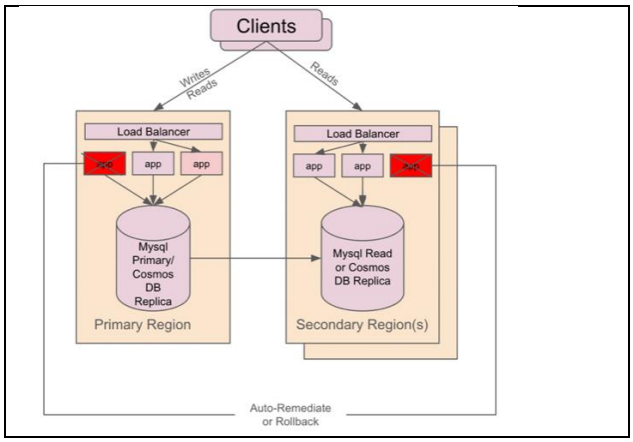

Figure 2: Summarized mitigation for global app failures

B. Recovery from regional application failures

In this scenario, we assume that applications are supported by globally replicated databases, which may become inaccessible within a region due to either regional outages or problematic software deployments impacting the entire region. The following combined approaches have demonstrated comparative effectiveness in addressing such failures:

1) ***Rollback:***Infrastructure as Code (IaC) is especially valuable in this context. It is strongly recommended that frontline operational teams adopt IaC if not already implemented. Industry-standard

tools such as Terraform facilitate automated recovery from faulty code deployments by enabling rapid rollback of infrastructure changes.

2) ***Load Shifting to the Application Tier in a Healthy Region:***Two main strategies are commonly employed. The first involves redirecting client traffic to a secondary application tier while maintaining connectivity to the database in the primary region, thus preserving the original deployment. This configuration can be reverted with minimal cost once the primary region recovers. The second strategy shifts the entire application stack to the secondary region. In cases involving databases without global write replication—such as Azure Database for MySQL—this approach requires explicit provisioning of new write replicas, which is both costly and time-consuming, resulting in increased mean time to recovery (MTTR). Consequently, globally writable databases like Azure Cosmos DB are preferred. These strategies are summarized in Figures 3 and 4.

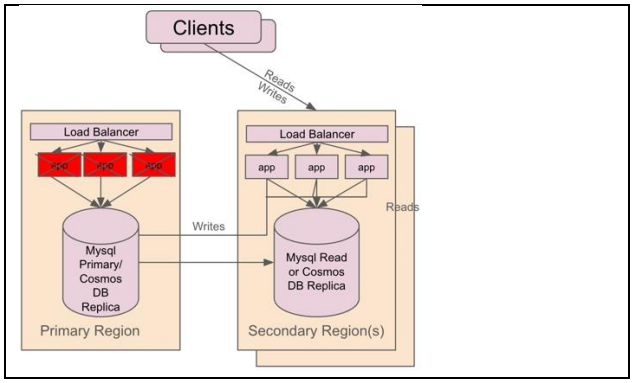

Figure 3: Approach 1: Traffic shift only for application instances

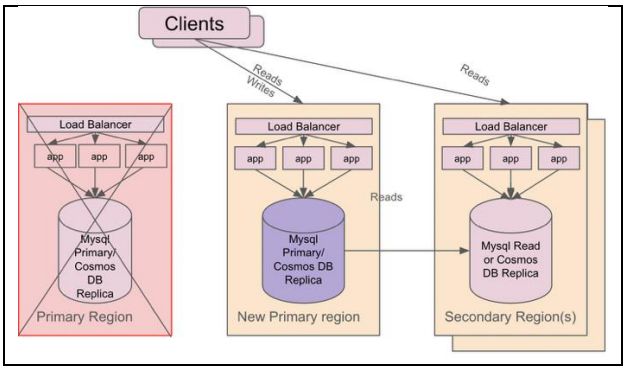

Figure 4: Approach 2: Traffic shift only for entire stack (expensive if database is intact)

3) ***DNS Record Updates:***In the event of regional outages, applications backed by multi-region write-enabled databases benefit from a simplified failover process. This typically involves updating DNS records to redirect traffic to the load balancer in a healthy secondary region, allowing the application tier in that region to serve requests while the affected region undergoes recovery. However, the applications considered in this section are limited to single region write configurations.

4) ***Configuration Validation:***Read replicas are often enabled through specific configuration flags that may require updates—either online or offline—to ensure seamless promotion of a secondary region to primary status. Verifying and applying these configuration changes is critical to allow the new primary instances to handle both read and write operations correctly.

C. *Recovery from non-regional database failures*

Assuming the application remains operational while the storage layer experiences downtime due to outages or data corruption, maintaining database replication and backups becomes critical for ensuring availability and resilience. For instance, distributed databases such as Azure Cosmos DB maintain multiple replicas—typically four within a region—distributed across different availability zones. This configuration enables automatic handling of non-regional outages through geographic replica placement.

In other architectures, where a primary write replica synchronizes data to secondary regions, recovery often requires provisioning a new database server in the affected region. Take managed services like Azure Database for MySQL, for instance. If something goes wrong—like a hardware failure or a network issue—it automatically spins up a new server to keep things running smoothly. Likewise, Azure Storage keeps multiple copies of your data (usually three) so that even if one goes down, your data stays safe and available. Once a new server is provisioned, remote storage is automatically attached without requiring manual intervention. From an operational standpoint, this process is seamless; however, any uncommitted transactions during the failure window may be lost and must be retried by the application. Such managed database services typically guarantee a service-level agreement (SLA) of 99.99% availability in scenarios involving non-regional outages.

D. *Recovery using a DR Region*

A traditional approach to disaster recovery (DR) involves establishing a secondary DR region configured to receive write traffic only in the event of a failover, while remaining offline during normal operations. Although this method provides a clear recovery path, it is often cost-prohibitive and challenging to maintain a full replica of the entire application stack. Additionally, continuous validation through replicated production traffic is required to ensure readiness.Consequently, maintaining a predominantly offline DR region is generally considered suboptimal due to its complexity and operational overhead.

E. *Automated Failovers*

During an outage affecting the primary write region, implementing an automated failover strategy based on a predefined action plan is essential. For example, if an outage is caused by too much traffic, the best move is to spread the load across other healthy regions. But if the issue is something more serious—like storage corruption or a full system failure—it makes sense to switch everything over to a backup region. In setups with multiple regions, it's smart to plan ahead by setting failover preferences. That way, traffic gets routed to the closest working region first, helping to keep latency low and the user experience smooth.

XII. MULTI-REGION APPLICATIONS MULTI-REGION WRITE AND MULTI-REGION READS

This section addresses disaster recovery strategies for applications designed to handle both write and read operations across multiple regions.Such applications typically follow a deployment model in which application instances—commonly web services—process write (PUT/POST/DELETE) and read (GET) requests in both primary and secondary regions. The secondary region may function as a disaster recovery (DR) site or as an additional active region serving traffic. The underlying database layer often employs globally replicated storage, supporting either read and write operations within the same region or write operations in one region with reads served from others.

A simplified representation of this architecture is depicted in Figure 5.

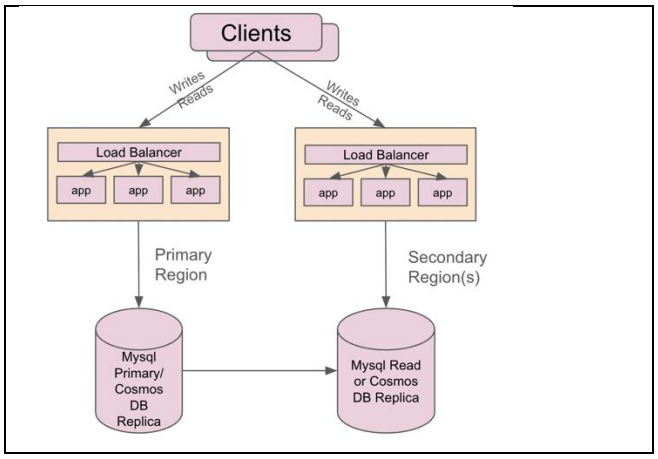

Figure 5: Multi-Region Read and Write Setup

The recovery strategy for regional and non-regional application failures aligns closely with that of multi-region applications configured for single-region writes and multi-region reads. Similarly, recovery from non-regional database failures using globally replicated databases, such as Azure Cosmos DB, follows comparable procedures.

However, differences arise in the context of regional database failures. Applications leveraging databases with built-in global replication—such as Azure Cosmos DB—benefit from inherent disaster recovery capabilities. Conversely, for databases that do not support multi-region writes, local writes must be performed, followed by synchronization of state across regions using consensus protocols. This approach prioritizes low write latency and is well-suited for use cases such as cloud providers managing asynchronous data uploads across devices. In contrast, industries requiring strong consistency guarantees—such as banking—generally avoid this approach, as it places the burden of consistency enforcement on the application layer.

XIII. Multi-Region applications Single Region Write and Single-Region Read

This section focuses on disaster recovery strategies for applications that handle both read and write operations within a single region. Such architectures are not designed for multi-region disaster recovery and therefore require robust local, region-specific recovery mechanisms to ensure continued operation.

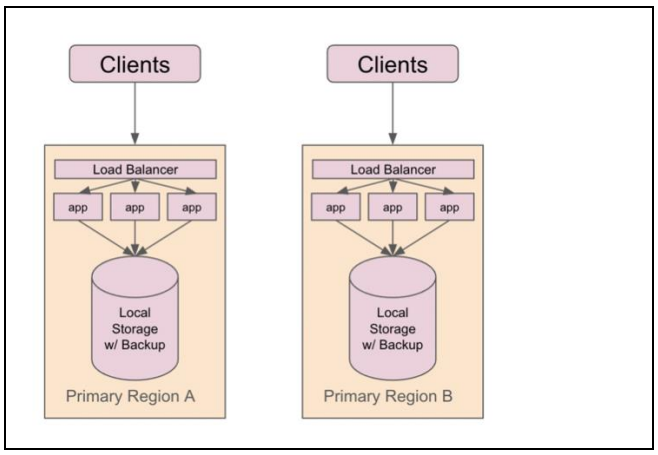

Figure 6: Single Region Write and Read Setup

A. Recovery from Regional Application Failures

If you can fix the issue quickly by rolling back to a previous version, it's best to use Infrastructure as a Service (IaaS) tools to do that efficiently. But when it comes to configuration changes, it's usually better to roll forward—just like teams often do in traditional on-premises setups—by applying a new, corrected change rather than going backward.

B. Recovery from Non-Regional Database Failures

If database corruption happens in one region it is a good idea to keep a read-only online replica in the same (primary) region. Offline backups are encourages in case of multiple failovers. If something goes wrong with the main database, it is a common practive to quickly promote this replica to take over, so it can handle both reading and writing without major disruption. It might be as easy as using CNAME records for databases and then updating these DNS records to point to the available replica.

C. Recovery from Regional Database Failures

In architectures where applications perform read and write operations exclusively within a single region and lack read replicas in other regions, a regional outage results in complete data layer unavailability. Recovery in such scenarios requires operators to restore data from backups, causing application downtime and potential data loss. It is therefore recommended to configure zonal redundancy within the same region at a minimum to enhance resilience.

XIV. Sample Recovery plan

The control plane responsible for managing multiple applications and storage instances requires a phased approach for effective disaster recovery.

A. *Phase 0: Preparation and Assessment of Disaster Recovery Strategies*

This paper has outlined various DR strategies corresponding to different architectural models. The initial step involves compiling an inventory of all applications managed by the control plane, assessing their current DR readiness status, and mapping them to applicable recovery strategies.

B. *Phase 1: Collaboration with Application Owners to Develop Runbooks*

Following Phase 0, it is essential to engage with application owners through a coordinated initiative, which aligns with industry best practices. This phase includes:

1) Identifying the existing architectures of control plane applications and transitioning them toward recommended architectures.
2) Developing comprehensive documentation that details the recommended architectures and the rationale behind their selection.
3) Establishing a centralized tracker to monitor application readiness and reference documentation.
4) Creating a standardized runbook per architecture to guide manual failover procedures during disaster scenarios.

C. *Phase 2: Validation of Disaster Recovery Regions for Control Plane Services*

In production environments, fault-tolerant zones—often distributed across multiple geographic regions or pseudo-regions—are established incrementally. Typically, a single control plane manages multiple application or data planes. For services conforming to recommended architectures and capable of handling cross-region traffic, the runbooks developed in Phase 1 serve as the operational guide to initiate failovers. This phase requires prior completion of Phase 1 to ensure architectural readiness.

D. *Phase 3: Development of Automated Tooling for Disaster Scenarios:*

The subsequent phase involves designing and implementing automation tools to facilitate failover testing at both region and service levels. Site Reliability Engineers (SREs) and operations teams lead efforts to automate the runbook procedures, with validation conducted initially in testing environments and subsequently in production. This phase also includes:

1) Defining Service Level Objectives (SLOs) and Service Level Indicators (SLIs) for failover performance.

2) Developing comprehensive tooling capable of executing scheduled failover tests on a quarterly or semi-annual basis.

E. *Phase 4: Organization-Wide Adoption of Disaster Recovery Tooling*

Once validated, the automated tooling is promoted and adopted across the entire control plane to ensure consistency and reliability in DR processes.

F. *Phase 5: Establishment of Periodic Failover Testing Plans*

The final phase involves collaboration with application owners to schedule and execute automated failover tests periodically, ensuring disaster recovery plans remain current and effective over time.

CONCLUSION

Based on the preceding analysis, we propose the following recommendations:

1) Multi-region applications supporting both multi-region write and multi-region read operations, as well as those with single-region write and multi-region read capabilities, should utilize globally distributed storage solutions such as Azure Cosmos DB.

2) Multi-region applications configured for single-region write and single-region read operations are well-suited to employ managed database services like Azure Database for MySQL.

3) Globally distributed storage systems, exemplified by Azure Cosmos DB, may also be considered for multi-region applications with single-region write and read patterns, depending on specific use cases.

4) Locally deployed storage solutions, such as standalone MySQL instances, can be acceptable for multi-region applications with single-region write and multi-region read architectures; however, migration to globally distributed storage options is strongly advised at the earliest opportunity.

REFERENCES

[1] Microsoft Azure, *Azure Database for MySQL*. [Online]. Available: https://learn.microsoft.com/en-us/azure/mysql/. [Accessed: 17-May-2025].

[2] Microsoft Corporation, *Azure Cosmos DB documentation*, Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/azure/cosmos-db/. [Accessed: 17-May-2025].

[3] Microsoft Azure, *Virtual Machine Scale Sets*. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/. [Accessed: 17-May-2025].

[4] Microsoft Azure, *Azure virtual machine automatic remediation*. [Online]. Available: https://learn.microsoft.com/en-us/azure/automanage/automanage-virtual-machines/. [Accessed: 17-May-2025].

[5] Facebook Engineering, "Under the Hood: Automated Backups," Facebook Notes, 2013. [Online]. Available: https://www.facebook.com/notes/facebook-engineering/under-the-hood-automated-backups/10151239431923920/. [Accessed: May 17, 2025].

[6] Amazon Web Services, "Disaster Recovery Options in the Cloud," AWS Whitepapers, 2023. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html. [Accessed: May 17, 2025].

[7] Amazon Web Services, "Disaster Recovery Options in the Cloud," AWS Whitepapers, 2023. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html. [Accessed: May 17, 2025]