Consistent Access to Cloud Services across Regions for Large Enterprises

Pavvan Pradeep
Computer Science and Engineering Department
PES University
Bengaluru, India
e-mail: pavvanpradeep@gmail.com

Aditi Srinivas M
Computer Science and Engineering Department
PES University
Bengaluru, India
e-mail: aditimatti@gmail.com

Prisha Goel
Computer Science and Engineering Department
PES University
Bengaluru, India
e-mail: prishapgoel@gmail.com

Dhruv Sanjaykumar Ratanpara
Computer Science and Engineering Department
PES University
Bengaluru, India
e-mail: dhruv2502@gmail.com

Shilpa S
Computer Science and Engineering Department
PES University
Bengaluru, India
e-mail: shilpas@pes.edu

***Abstract*—In today's world, high availability is critical to meet the needs of uninterrupted operation and customer satisfaction. The expansion of cloud services has permitted substantial progress towards reaching this availability. However, there are issues such as high ownership costs and constant availability across many locations, as not all cloud providers provide comprehensive regional support. This project seeks to demonstrate a strategy for developing platforms for global organizations that are available in multiple regions and provide an improved user experience. Our methodology enables one to seamlessly integrate the Istio service mesh into the existing infrastructure, with a focus on high performance,low latency, multi-region availability across many zones, dispersed deployment for better reliability, and a low total cost of ownership. Our solution uses Istio's features to improve service resilience and distribution, resulting in cost-effective, high-performance multi-region deployments.**

***Keywords*—Istio Service Mesh; Reduced total cost of ownership; Multi-region failover; Availability Zone level failover; Minimized latency.**

I. Introduction

High availability, scalability, and scattered deployments are vital in today's technology-driven world, where continuous access to services is essential [1]. Cloud providers such as Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, and Oracle Cloud Infrastructure (OCI) have played significant roles in providing these services with multi-region availability, ensuring that operations continue even when regions fail. AWS is the largest cloud provider, noted for its massive infrastructure, which includes 48 regions and 54 Availability Zones worldwide, with a range of services including processing power, storage, and databases. Cloud providers like AWS use auto-scaling mechanisms to maintain availability during periods of high demand [2]. While each operator provides a variety of global regions and availability zones, achieving seamless worldwide coverage remains a difficulty. Geographic limitations, high ownership costs, and interoperability between providers remain significant difficulties. Most of the industries are moving towards a microservices architecture for their applications to enhance availability. Deploying applications as lightweight portable container enhances scalability and fault-tolerance [3]. However, enterprises frequently struggle to balance high availability needs with cost-effectiveness, particularly when deploying across various geographies [4]. There is also the challenge of managing dispersed applications, data consistency, and compliance requirements, especially when apps cross international borders. Existing cloud-based high-availability solutions face several limitations. Many rely heavily on cloud providers, leading to vendor lock-in and high operational costs. Cloud-based auto-scaling guarantees resource availability, but it falls short in addressing region-level failover. It can be challenging for businesses to maintain a uniform infrastructure around the globe because some cloud services are not accessible everywhere. Given these difficulties, a method that minimizes reliance on the cloud, maximizes latency, and guarantees smooth failover without compromising cost is required.

Our approach involves a one-time infrastructure setup cost, after which it leverages open-source tools like Istio and a custom load balancer to enable seamless failover and minimize latency. Our aim is to integrate the Istio service mesh into large organisations' existing architecture for enhanced traffic routing and service maintenance. This approach aims to improve availability, provide multi-region resilience, and reduce ownership costs. Our technique routes traffic to the nearest region to reduce latency and ensures that users receive responses within an optimal time. When a service in a given availability zone goes down, the zone is tagged as unhealthy, blocking routing of further user requests to it. This feature helps to avoid inter-zone service communication, which might increase latency.

Our approach uses locality-based load balancing to evenly distribute traffic across all healthy zones within a region. Such a balanced distribution ensures stability and prevents services in any zone from being overloaded with user requests thereby

increasing total system resilience. Our technique uses the open-source Istio service mesh to manage traffic effectively. By using Istio, we can create a strong architecture with greater stability, availability and scalability while lowering the total cost of ownership.

Our approach focuses on improving the private infrastructure of large-scale industries. It integrates easily with their existing systems to boost availability, reduce latency, and lower costs by reducing reliance on cloud providers. Since cloud services are not available in all regions and don't always work well together, relying on cloud services completely is not a good option for large-scale industries. Moreover migrating applications to the cloud require one to have many procedures in place such as application migration, data migration and dependency checks [5]. Instead large-scale industries can set up their infrastructure in the regions they need and integrate our approach seamlessly to increase the reliability and availability of their services across multiple regions, making their systems more efficient and cost-effective. Hence, our approach is a solution that meets the needs of modern, large-scale industries.

Section 2 presents a literature survey, exploring current strategies aimed at minimizing latency and increasing multi-region availability. The methodology is described in Section 3, along with the technologies utilized, such as the Istio service mesh and Kubernetes in Docker (KinD). This section explains the implementation of our setup, which consists of two clusters and a load balancer to ensure multi-region availability and optimized latency. Results are presented in Section 4, along with our framework's latency measures. The research is finally concluded in Section 5, as part of the future scope the paper suggests use of AI models for fault prediction and resource optimization to enhance system maintenance.

II. Literature Survey

A. Malhotra, A. Elsayed, R. Torres and S. Venkatram [6] explore solutions to achieve near-zero downtime for cloud-native, business-critical applications. Thw authors emphasize on microservices architecture to enhance fault tolerance and scalability. A clustered setup with Kubernetes enables load balancing and seamless failovers within and across regions. Tools like Global Load Balancer (GLB) for geo-based routing, PgBouncer for PostgreSQL optimization, and HAProxy for high availability are integrated. For read-heavy applications, read replicas handle most requests to offload the primary database, while write-heavy setups distribute traffic geographically to optimize latency. Despite its robust approach, the architecture focuses heavily on database resilience, with limited attention to network-level failovers.

A. Anwar's [7] study proposes a high-availability solution leveraging AWS services. Applications are hosted on EC2 instances spread across multiple Availability Zones (AZs) with a load balancer managing traffic distribution. NAT Gateways ensure secure internet connectivity, while auto-scaling groups dynamically adjust resources. Elastic IPs provide stability, maintaining consistent DNS entries during instance restarts. Although this setup withstands zonal outages and high traffic, regional outages remain a challenge. Its reliance on AWS services also leads to high operational costs, which could be optimized by refining auto-scaling policies.

Anna Berenberg and Brad Calder [8] evaluate different deployment models, from zonal to global, highlighting their benefits like increased availability, improved latency, and scalability. By integrating edge computing, it suggests further latency reductions and resource optimization. While the archetypes offer flexibility and future-proofing, managing complex deployment models and addressing global outages pose challenges. Balancing costs with high availability and latency remains an ongoing issue in these strategies.

A. Hajikhani and A. Suominen [9] focus on disaster recovery, this research outlines strategies for applications across Kubernetes clusters using service mesh and serverless workloads. It emphasizes automatic failover mechanisms, resource optimization, and cost reduction in multi-cluster environments. While the paper provides practical insights, it falls short in covering all failure scenarios and complex multi-cluster deployments. Its narrow focus on Kubernetes limits its broader applicability to other cloud setups.

Mohammad Reza Mesbahi, Amir Masoud Rahmani and Mehdi Hosseinzadeh [10] present a roadmap for achieving high availability and reliability in cloud environments. The paper identifies challenges and proposes solutions to align with quality-of-service agreements. While insightful for both providers and consumers, the paper lacks practical validation and oversimplifies complex technical issues. It serves as a theoretical framework rather than a hands-on guide to cloud resilience.

Our proposed architecture aims to address the challenges faced by these previous approaches by removing the dependency on existing cloud providers, and minimizing cost and latency.

III. Methadology

Our methodology is designed to leverage a series of open source technologies, to ensure multi-region availability, efficient load balancing, and reduced latency.

A. Kubernetes in Docker(KinD)

KinD (Kubernetes in Docker) is a tool that allows one to easily establish and manage Kubernetes clusters on a local workstation using Docker containers. Kind is intended to simplify Kubernetes development and testing by enabling:

- Experimenting with various Kubernetes versions, configurations, and deployments.
- Simulation of a multi-node Kubernetes setup on a single PC.

B. Istio Service Mesh

A service mesh is a software layer that facilitates communication between services in an application. It consists of a network of proxies known as "sidecars" that run alongside each service. A service mesh can help prevent cascade failures, which can cause system-wide downtime. It accomplishes this

by including features such as circuit breaking, retries, and timeouts. A service mesh can monitor the status of services, connection, and failures. It can also deliver metrics, logs, and trace data. A service mesh can aid with traffic management, such as load balancing and rate limitation. A service mesh is independent of each service's code, allowing it to function across network boundaries and with various service management systems.

An Istio service mesh is open source and functions as a dedicated infrastructure layer that manages and secures communication between microservices within a distributed application, effectively giving a transparent mechanism to govern traffic flow, security, and observability.
Key elements of the Istio architecture:

- The Data Plane:
  The Data Plane consists of "Envoy" proxy sidecars that handle network traffic, routing, load balancing, and telemetry data collection for each microservice.
- Control Plane (Istio):
  A centralized control system for configuring sidecar proxies, creating traffic routing rules, security policies, and other service mesh operations.

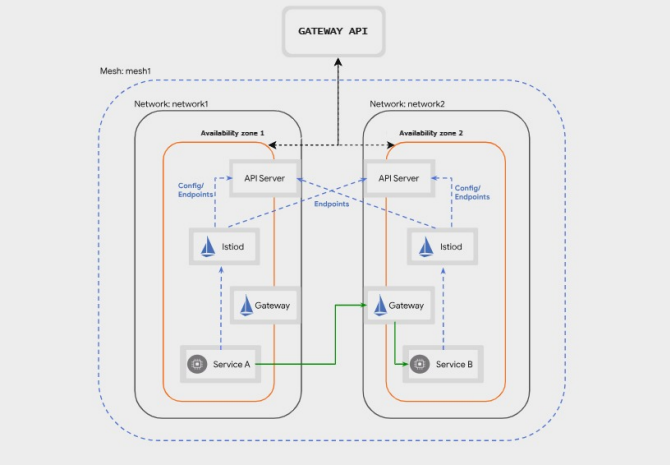

Figure 1. Services running across Availability Zones within a region

C. Multi Primary Istio on Different Networks

The multi-primary Istio configuration across various networks allows two clusters to run with independent control planes, resulting in secure and highly available communication across boundaries. Figure 1 illustrates our multi-primary Istio architecture within a region where there are two clusters on separate networks, each with its own Istio control plane. To support cross-cluster connectivity, both clusters have an Istio east-west gateway in place. This gateway manages east-west traffic for internal and cross-cluster service interactions, allowing ingress and egress traffic to flow smoothly between clusters.

To enable proper service discovery, each cluster's API servers are configured to recognise one another. This is performed by establishing a remote secret key in each cluster, allowing for mutual discovery and secure communication across networks. As a result, cluster 1's API server will be able to access cluster 2's services along with those within its own clusters, and vice versa. This configuration allows the Istio service mesh to effectively distribute traffic across availability zones, providing resilience, fault tolerance, and effective load balancing inside the service mesh.

D. Load Balancer

A Load balancer is used to distribute user requests to the under-loaded services to ensure that no service is overwhelmed with requests. Hence, it is necessary to have a fault-tolerant load balancer to create a highly resilient architecture [11]. If the load is not distributed efficiently across all the available services it degrades the performance and efficiency of computing resources [12]. Our architecture leverages a global load balancer to achieve high availability and optimal performance by routing traffic across two geographically separated regions. The primary goal is to direct users to the nearest healthy instance, thereby minimizing latency and enhancing the user experience. This approach is essential for business-critical applications, as it ensures that users can access the service with minimal delay and without disruption, even in cases of regional failures. To achieve accurate location-based routing, we employ an external API that provides the user's geographical data based on their IP address, which is a reliable method for real-time location determination in distributed systems.

The process begins by constructing a URL with the user's IP address, which is then used to make an HTTP GET request to retrieve the user's location data. From the JSON response body, we parse the latitude and longitude values, which serve as input for calculating the distance to each available instance. Before proceeding with distance calculation, we must first verify the health of each instance to avoid directing traffic to an unavailable or unstable service. For this purpose, we create a health check URL by formatting each server's address and port, which acts as an endpoint to verify its availability. This validation step is crucial, as it prevents unnecessary errors and guarantees that only active and responsive instances are included in the load balancing process. Each server URL is parsed to ensure validity; if it is found invalid, the server is marked as unhealthy, and an error message is logged. To further ensure robust operations, we use a mutex lock, which prevents race conditions by managing concurrent access to shared resources during health checks, a common practice for maintaining consistency in multi-threaded applications.

Once the health of each instance is confirmed, we convert the latitude and longitude coordinates of both the user and each healthy instance into coordinates suitable for distance calculation. To accurately measure the distance between these points, we utilize the Haversine function. This mathematical formula is specifically designed for calculating the shortest distance between two points on a sphere, making it highly effective for geographic distance calculations. The precision of the Haversine function is beneficial for our load balancer,

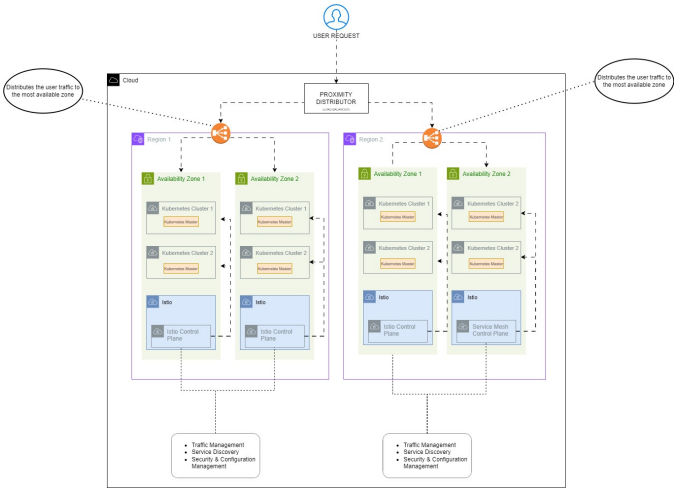

Figure 2. Architecture

as it ensures users are routed to the physically closest instance, which is particularly important in regions with multiple service points.

After calculating the distances, the global load balancer routes traffic to the closest healthy instance, significantly reducing latency and providing users with a faster, more reliable service. In the event that this instance becomes unavailable due to a failure or outage, the load balancer dynamically reroutes traffic to the next closest healthy instance, maintaining continuous availability. This resilient approach to load balancing not only enhances user satisfaction but also aligns with best practices in high-availability architecture by reducing single points of failure and ensuring reliable service continuity across regions. Figure 2 depicts a high level overview of the entire architecture of our setup. By leveraging a multi-cluster Kubernetes setup with Istio, it enhances service discovery and traffic management across all regions. This holistic approach to existing infrastructure of companies guarantees high availability, scalability, and optimal performance for enterprise applications.

E. Our Setup

Our setup is distributed across two regions, each containing two Kubernetes clusters that serve as Availability Zones within those regions. Such an architectural setup ensures high availability and low -latency. These clusters are managed using KinD (Kubernetes in Docker). The regions are labeled as region1 and region2, and the Availability Zones within each region denoted as zone1 and zone2. Figure 3 illustrates our proposed architecture. This setup enables seamless failover and efficient workload distribution across regions. It also helps in mitigating single points of failure by ensuring redundancy at both the regional and availability zone levels. Additionally, traffic routing mechanisms are implemented to direct user requests to the nearest and most responsive cluster, improving overall performance.

Istio is installed in a multi-primary configuration on each cluster. This configuration ensures high availability and functionality by providing each cluster with its own Istio control plane. Each cluster in a multi-primary setup has its own Istio control plane, which allows them to govern traffic, implement rules, and forward requests throughout the mesh. Such a setup ensures system stability and availability. Even if the control plane of one of the clusters fails, the other clusters remain operational and continue to serve the user requests.

The helloworld service is deployed across all clusters, providing redundancy and locality-based access. The service instances are labeled according to their region and zone easier traffic routing:

- The service is labeled as helloworld.region1.zone1 in region1, zone1 and helloworld.region1.zone2 in region1, zone2.
- Similarly, the service is labeled as helloworld.region2.zone1 in region2, zone1 and the service is labeled as helloworld.region2.zone2 in region2, zone2 .
- The helloworld gateway allows access to the services.

To ensure minimal latency and efficient resource utilization, a custom load balancer is deployed on a separate system. This load balancer routes user requests to the nearest region, optimizing performance and reliability.

F. Availability Zone Failover using Istio Locality-Based Routing

For enhanced resilience, Istio's locality-based failover is configured to handle availability zone-specific issues seamlessly. If an issue arises within any zone, marking it unhealthy, Istio's destination rules redirect all traffic to the nearest healthy zones within the same region. This immediate redirection ensures that users experience minimal disruptions, as the system reroutes requests automatically to healthy zones, continuing services which makes the architecture highly available. Highly available systems are designed so that no single failure causes unacceptable service disruption [13]. The destination rule configuration also continuously monitors the health status of each zone. Once the affected zone is restored and services are healthy, it becomes eligible again to receive requests, maintaining balanced and resilient service distribution.

G. Region-Wide Failover for Regional Resilience

In cases of region-level failures, the custom load balancer takes over to prevent routing to any impacted region. It continuously performs health checks across regions to ensure only healthy regions handle requests. If a region fails, the load balancer seamlessly routes traffic to the other available region, ensuring uninterrupted service. As soon as all services in the affected region are confirmed to be operational again, the load balancer resumes routing requests to it.

IV. Results and Discussion

In our observability setup for Cluster 1 in Region 1, Zone 1, we monitored key performance indicators such as latency,

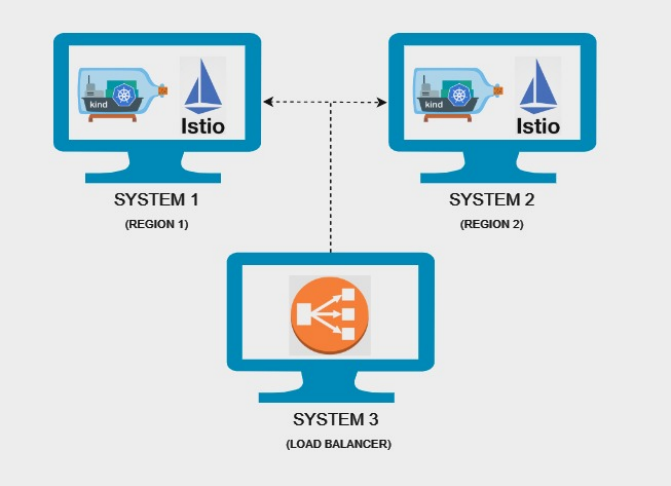

Figure 3. Our setup

success rates, and request volumes. Here are the detailed observations:

A. Traffic Volume and Success Rate:

As seen Figure 4 the incoming traffic volume is 12.2 requests per second, with a recorded success rate of 100%. This indicates that every request was processed successfully, without any loss or errors, reflecting a highly reliable service operation.

Figure 4 shows that both 4xx and 5xx error rates are recorded at 0, showing there were no client-side or server-side errors. The absence of 4xx errors suggests that all client requests were well-formed and valid, while the lack of 5xx errors confirms robust server-side processing and stability.

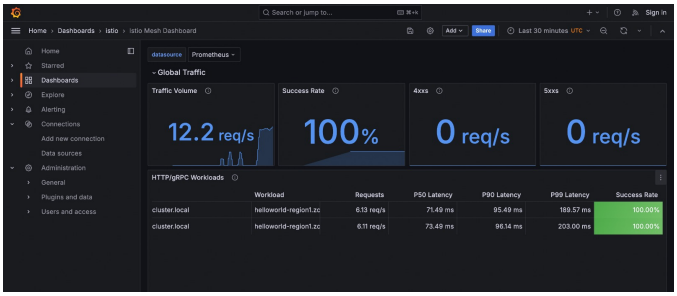

Figure 4. Observability Metrics

B. Latency Metrics (P50, P90, P99):

Since the helloworld service in Region 2, Zone 2 is also accessible from Region 1, Zone 1, we compared latency across both clusters:

P50 Latency: Figure 4 shows that the median latency, or time within which 50% of requests are processed, is 71.40 ms for Cluster 1 (Region 1, Zone 1) and 73.49 ms for Cluster 2 (Region 1, Zone 2).

P90 Latency: It seen in Figure 4 that the latency within which 90% of requests are completed is 95.49 ms in Zone 1 and 96.14 ms in Zone 2, showing that the majority of requests have relatively low latency.

P99 Latency: Figure 4 shows that for 99% of requests, the latency is 189.57 ms in Zone 1 and 203.00 ms in Zone 2. These values show that even for high-percentile latency, processing times remain well within acceptable ranges.

The slight increase in latency in Zone 2 (Region 1) is attributed to cross-zone access, as requests from Zone 1 accessing services in Zone 2 experience a maximum additional latency of approximately 4 ms.

C. Client and Server Request Volumes and Success Rates:

The client request volume is observed to be 6.4 operations per second (ops/sec), matching the server request volume of 6.4 ops/sec. This consistency indicates that all requests initiated by the client are successfully received and processed by the server, with no data loss or retries.

Both client and server success rates are recorded at 100%, further confirming the absence of failures or unfulfilled requests.

D. Client Request Duration:

From start to finish, client queries typically take around 300 milliseconds. This is the total round-trip time, which includes processing requests, creating responses, and returning them to the client.

E. Gateway Success Rate:

The helloworld gateway's incoming request success rate is 100%, which shows that the gateway setup is operating correctly and consistently, directing traffic without introducing issues.

F. Outgoing Response Rate by Destination Workload:

The destination workload's outbound response rate is measured at 100%, indicating that answers from the destination workload are constantly successful, indicating error-free communication between workloads and optimum processing of outgoing data.

All the metrics point to a robust and highly available system with low latency for both local and cross-zone queries. The multi-region setup can process requests at low latency and ensure that the system is highly responsive. Such a highly available architecture is needed for large enterprises to ensure service uptime and an enhanced user experience. Including observability into the infrastructure using tools like grafana enables enables real-time tracking and timely identification of potential issues.

Such high success rates and low latency values confirms that the setup is configured to withstand traffic spikes and variations without any effect on service quality, thereby emphasizing on its effectiveness in a multi-zone, multi-region architecture.

G. Cost Analysis:

For production-grade servers across regions (e.g., T3.medium or m5.large instances), assuming 10-20 instances per region for redundancy, costs could range from $1,500 to $3,000 per region, totaling $3,000 to $6,000. AWS EKS is priced at $0.10 per hour per cluster along with EC2 costs for worker nodes, could add approximately $1,000 to $2,000 for a multi-region setup. Multiple load balancers and high outbound data transfers may contribute an additional $500 to $2,000, while persistent storage with high availability (e.g., Amazon RDS or S3 for data durability) might cost around $500 to $1,000. Overall, for a mid-to-large-scale deployment, the monthly operational cost might range from $5,000 to $12,000. If large corporations and real-time applications were to implement an existing architecture like this, their costs could increase to anywhere between $20,000 to $50,000 monthly. We aim to target large scale industries who have an existing private infrastructure.By leveraging open-source tools and technologies and avoiding reliance on AWS or any other cloud providers, the total cost of operation for our proposed methodology comes up to $0.

H. Comparison with Cloud-Native Solutions:

Cloud providers such as Google's Anthos and AWS EKS with Istio offer managed service mesh solutions. Our method, leveraging KinD and Istio in a multi-primary setup, eliminates cloud dependency while maintaining low-latency service discovery and failover. Some enterprises, like Cloudflare and AWS CloudFront, deploy edge nodes or use CDNs to reduce latency. Our approach dynamically routes traffic based on region health and network proximity, reducing costs while ensuring resilience. AWS Global Accelerator, GCP Multi-Region Load Balancer, and other providers offer auto-scaling groups, managed load balancers, and cross-region replication. While these services ensure failover and high availability, they introduce vendor lock-in and high operational costs.

V. Conclusion and Future Work

Our solution presents a robust, multi-region architecture in which there are two separate geographical regions which host independent Kubernetes clusters on KinD with Istio installed using the multi-primary approach on different networks. Each cluster is set up with a Hello World service. Each cluster has its own control plane, ensuring independent access across clusters. Such an architecture increases the availability of the service mesh and ensures that failure in one cluster or region does not affect the other clusters, supporting high availability throughout the architecture.

Our setup uses a load balancer that directs traffic based on real-time health monitoring. Every 10 seconds, the load balancer checks the health of each cluster to ensure that traffic only goes to healthy clusters. If a cluster fails, the load balancer identifies the nodes as unhealthy and routes traffic to other healthy nodes in the service mesh. This strategy provides constant uptime and lowers latency, resulting in a smooth user experience even amid infrastructure failures.

Cloud providers like AWS provide ways to design fault-tolerant cloud application using virtualization technologies which emphasize on redundancy and continuous monitoring [14]. However, using such approaches can be very expensive for large-scale industries. Our approach is especially useful for such large-scale businesses who want to optimise their infrastructure and reduce reliance on expensive cloud alternatives. Our setup is a robust, multi-region configuration which leverages open-source solutions like Istio, to deliver multi-regional availability, low latency, and scalability without incurring the significant expenditures associated with cloud services. This design seeks to integrate smoothly with current organisational environments improving resilience and service reliability across many regions. By using open-source solutions, we provide a low-cost way to achieve a distributed, highly available service architecture. This makes our method ideal for large-scale organisations that value service continuity, scalability, and low-latency access for worldwide users.

Integrating powerful AI-powered monitoring capabilities could be one of the ways to improve the infrastructure. By implementing machine learning models, the system can predict faults earlier and optimise resource allocation. Such an approach might provide an additional layer of maintenance by detecting flaws before they impact operations, decreasing downtime and enhancing system reliability, especially during peak demand periods.

Reducing latency across multi-region architectures, particularly during region-wide or availability zone failovers, can improve user experience. Response times could be reduced by using enhanced routing algorithms and advanced load-balancing approaches. By lowering latency, the infrastructure may provide a better user experience while maintaining high availability and performance. This would provide seamless access to services, resulting in a more resilient and efficient system.

References

[1] A. Malhotra, A. Elsayed, R. Torres, and S. Venkatraman, "Evaluate Solutions for Achieving High Availability or Near Zero Downtime for Cloud Native Enterprise Applications," in *IEEE Access*, vol. 11, pp. 85384-85394, 2023, doi: 10.1109/ACCESS.2023.3303430.

[2] A. Johnson and B. Lee, "Auto-Scaling Strategies for High Availability in AWS," in *Proceedings of the International Conference on Cloud Computing*, 2017, pp. 112-125.

[3] H. Lang and C. Li, "Containerization for High Availability in Cloud Environments," in *Proceedings of the International Conference on Cloud Computing and Big Data*, 2019, pp. 201-214.

[4] G. Verma and R. Sushil, *Cloud Computing Implementation: Key Issues and Solutions*, Springer, 2015.

[5] N. Ahmad et al., "Strategy and Procedures for Migration to Cloud Computing," in *Proceedings of the 2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, 2018, pp. 1-5.

[6] V. Mohammadian et al., "Fault-Tolerant Load Balancing in Cloud Computing: A Systematic Literature Review," in *IEEE Access*, vol. PP, pp. 1-1, 2021.

[7] W. A. Aziz, "High Availability Solution for Cloud Applications," in *International Journal of Simulation: Systems, Science Technology*, vol. 24, 2023.

[8] A. Berenberg and B. Calder, "Deployment Archetypes for Cloud Applications," in *ACM Computing Surveys*, vol. 55, no. 3, Article 61, pp. 1-48, March 2023. doi: 10.1145/3498336.

[9] A. Hajikhani and A. Suominen, "The Interrelation of Sustainable Development Goals in Publications and Patents: A Machine Learning Approach," *Trepo Digital Repository, Tampere University*, 2021.

[10] M. R. Mesbahi, A. M. Rahmani, and M. Hosseinzadeh, "Reliability and High Availability in Cloud Computing Environments: A Reference Roadmap," in *Human-Centric Computing and Information Sciences*, vol. 8, no. 20, 2018. doi: 10.1186/s13673-018-0143-8.

[11] A. Malhotra, A. Elsayed, R. Torres, and S. Venkatraman, "Evaluate Solutions for Achieving High Availability or Near Zero Downtime for Cloud Native Enterprise Applications," in *IEEE Access*, vol. 11, pp. 85384-85394, 2023, doi: 10.1109/ACCESS.2023.3303430.

[12] S. Afzal and G. Kavitha, "Load Balancing in Cloud Computing – A Hierarchical Taxonomical Classification," in *Journal of Cloud Computing*, vol. 8, no. 22, 2019. doi: 10.1186/s13677-019-0146-7.

[13] E. Bauer and R. Adams, "Service Reliability and Service Availability," in *Reliability and Availability of Cloud Computing*, Hoboken, NJ: Wiley-IEEE Press, 2012.

[14] R. Brown and M. Davis, "Designing Fault-Tolerant Cloud Applications: A Virtualization-Based Approach," in *IEEE Transactions on Services Computing*, vol. 18, no. 4, pp. 567-581, 2020.