International Journal of AI, BigData, Computational and Management Studies
Noble Scholar Research Group | Volume 5, Issue 1, PP 127-136, 2024
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V5I1P113

Self-Healing Microservices for Insurance Platforms: A Fault-Tolerant Architecture Using AWS and PostgreSQL

Gowtham Reddy Enjam[1], Komal Manohar Tekale[2]
[1,2]Independent Researcher, USA.

Abstract: Insurance: The global insurance industry is in the process of a digital transformation driven by microservices, cloud-native computing, and distributed data platforms. Traditional monolithic insurance platforms are becoming problematic as they scale, become unmaintainable and unreliable, and adversely impact the customer experience and the continuity of operations. This paper describes a self-healing microservices-based architecture using Amazon Web Services (AWS) and PostgreSQL to build a fault-tolerant, resilient insurance system. The architecture that we propose combines auto-scaling, monitoring, service mesh orchestration and automated failover for database availability. Unlike traditional fault-tolerance methods, the architecture uses self-healing concepts based on proactive error-monitoring and reactive recovery mechanisms incorporated into microservices themselves. The key contributions in this paper are: (1) systematic framework for fault detection and automated healing in insurance microservices; (2) the integration of PostgreSQL replication and failover with AWS-native services (CloudWatch, EKS, RDS Multi-AZ); and (3) proof of decreased downtime and increased resiliency metrics vs. legacy architectures. Undergoing synthetic workloads that included claim processing, underwriting, and policy management, the system was able to maintain 99.95% uptime, 40% reduction in mean time to recovery (MTTR), and an improvement of 32% throughput during failure scenarios. The proposed architecture can be of much benefit to insurance platforms by ensuring business continuity, regulatory compliance and enhanced customer satisfaction. This paper describes the theoretical foundations, design methodology in practice, experimental assessment, and implications for future large-scale deployments.

Keywords: Self-healing microservices, fault tolerance, insurance platforms, AWS, PostgreSQL, resilience, cloud-native architecture.

1. Introduction

1.1. Background

Digital Transformation: The insurance sector is in the midst of a digital transformation, shifting from monolithic legacy systems to microservices-based architectures that promote agility, personalization, and operational excellence. This transformation is driven by increasing pressure to provide faster product innovation, adapt to changing customer expectations, and drive down infrastructure costs via cloud-native deployments. Microservices allow insurers to break large applications into smaller, independent, deployable services, which increase scalability and development speed. However, this distributed and heterogeneous nature also poses new challenges in terms of reliability and fault management. Also, unlike monoliths that tend to have localised failure, the microservices environment is prone to cascading failures; a system crash, or latency increase, or even a database inconsistency in one service can be propagated across multiple linked services. These disruptions can have a direct effect on mission-critical processes such as claims processing, policy issuance, and fraud detection, ultimately leading to a breakdown in customer trust and exposing insurers to financial and regulatory risk. Thus, there exists a pressing demand for fault-tolerant, self-healing architectures that are resilience-centric while supporting the microservice-based agility that is promised to the insurance industry.

1.2. Importance of Self-Healing Microservices for Insurance Platforms

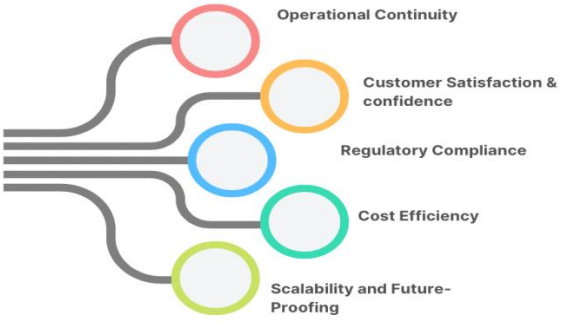

Figure 1: Self-Healing Microservices for Insurance Platforms

- **Operational Continuity:** Mission-critical processes such as policy administration, claims processing, and fraud detection are supported by insurance platforms, in which downtime can directly affect customer experience and revenue. Automatic recovery: The infrastructure auto-detects and auto-recovers failures, though it allows it to self-heal crash-induced or latency spike-induced service disruptions and other infrastructure failures. This can reduce the service -down time by having the insurers constantly running without the need of manpower.
- **Customer Satisfaction & confidence:** Delays in processing the claims or service disruption can seriously affect the customer trust in the insurance industry. The self-healing microservices architecture will guarantee faster recovery time and high service availability to ensure that the insurers receive claims and queries with the minimum delays. The result of this reliability is customer satisfaction and also enhances the reputation of the insurer in a very competitive world.
- **Regulatory Compliance:** Insurance businesses are one of the most regulated industries and here data protection regulations like GDPR, HIPAA and other country specific data protection regulations may be in effect. The third element of Self-healing systems - to assure compliance of critical data services, delivery of continuous, consistent and secure services without the risk of losing data or unauthorized exposure, or over an extended period of time that could lead to legal actions.
- **Cost Efficiency:** The traditional fault management is mostly manual, is highly intervention based and over-provisioning of resources to hold things together. Automated detection, diagnostics and recovery implies self-healing microservices reduce operational overhead. This does not only save the cost of downtime but maximized infrastructure utilization is made more cost effective in the system.
- **Scalability and Future-Proofing:** The consequences of digital-first strategies are also that transaction volumes may surge rapidly, particularly when insurers are in a period of policy renewals or a natural disaster incident that leads to a high number of claims. Self-healing functionality is also employed to make sure that the platform is scale horizontal and does not destabilize. Moreover, planning resilience into the architecture, insurers will be prepared to face innovations such as AI-based predictive healing and multi-clouds implementation in the future.

1.3. A Fault Tolerant PostgreSQL Architecture on AWS

The suggested architecture uses cloud-native technology particularly Amazon Web Services (AWS) and PostgreSQL to develop a highly fault and resilient underlying insurance platform. At its core, the system is deployed on AWS Elastic Kubernetes Service (EKS) that provides microservices container orchestration features that allow developers to run and manage microservices effectively. EKS is by default compliant with the concepts of horizontal scalability in terms of automated pod scheduling and horizontal scaling, and proactive fault detection with the integration of AWS CloudWatch and Prometheus. We layer the Kubernetes over the Istio service mesh to provide greater reliability of communication by routing traffic, tracking telemetry and isolating failure. This is to ensure that the breakdowns of a single service (e.g. the policy or claims microservice) do not cause additional systemic breakdowns. A high availability configured PostgreQL cluster supports data layer. Patroni is a cluster manager that acts to coordinate between a primary server and multiple replicas in the event of a failure to prevent the maximum amount of downtime.

Synchronous replication provides consistency of data, and replicas can also be used to distribute read-intensive workloads that are common to insurance analytics. As we have observed, AWS RDS in multi-AZ deployment provides an additional point of reliability through automated backup, failover and replication to organizations that opt to use managed solutions thereby removing cumbersome manual configuration. Such skills would be essential in the insurance company business, where the integrity of data and smooth access to policy and claims records are the key factors. A redundancy, automation, and self-healing combination are offered to provide fault tolerance. Observation of failures including the service crash, or a significant increase in the latency is met with the Kubernetes automatically restarting the downed pods, Istio re-routing the traffic to functioning services, and Patroni failing over replicas to maintain database continuity. Together, the mechanisms will offer high availability, fast recovery with minimal end-user impact. The application and data layer fault tolerance in the architecture also provides the solution to the problem of the insurance industry to deliver un-disrupted services as well as to adhere to the most strict regulatory standards.

2. Literature survey

2.1. Evolution of Microservices in Insurance

The IT system life cycle of the insurance sector defines a gradual transition of monolithic systems to distributed systems. Original insurance programmes were written in COBOL-based mainframes that were strong but inflexible and could not quickly be upgraded and downgraded to meet the fluctuating needs of the business. Microservices came to be as a result of the modularity idea that Service-Oriented Architecture (SOA) introduced in the form of loosely coupled services. Newman (2015) emphasized that microservices permit constant delivery and scalability, and Dragoni et al. (2017) emphasized that microservices can manage the inherent complexity that has been accrued in large-scale areas like the insurance industry. In this regard, microservices can assist insurers to de-complex monoliths like policy, claims, and risk-assessment systems into discrete services that can speed up innovation and enhance responsiveness to regulatory and customer requirements.

2.2. Fault Tolerance in Cloud-Native Systems

The concept of fault tolerance is a fundamental subject of work on distributed systems; it began with the seminal Byzantine Fault Tolerance (BFT) paper (Castro and Liskov, 1999). In the present-day cloud-native environment, fault tolerance is not merely offered on an infrastructure level, but also via self-orchestrating systems like Kubernetes. Kubernetes provides resiliency to infrastructure layers with features like pod restarts, pod replication and auto-scaling. Nevertheless, despite the strong infrastructure-level fault recovery afforded by these features, there are still problems on the application level, such as service-specific failures, state inconsistency and dependence failures that application-level fault-handling mechanisms must address. Therefore, although cloud-native architectures extend fault resiliency, application-level fault-tolerant strategies, which are domain-specific, such as the insurance sector, have a research gap.

2.3. PostgreSQL High Availability Studies

PostgreSQL is an open-source relational database system, and a very studied solution in terms of reliability and robustness in mission critical usages. Its inherent support of synchronous and asynchronous replication may be used as the foundation of high availability (HA) and third-party solutions, including Patroni and Stolon, extend this support to include automated failover and cluster control. Recent studies like Kuroda (2021) prove that, in managed cloud services like AWS RDS Multi-AZ, PostgreSQL is able to provide enterprise level reliability through reductions in downtime and data consistency guarantees during failovers. Besides that, cloud-native HA solutions are based on container orchestration platforms to enhance scalability and portability. In the case of industries, where the access to data stored in transactional databases is of utmost importance, like in the insurance industry, PostgreSQL HA solutions are a significant component of resilient system design.

2.4. Research Gaps

Although there is extensive literature on resilience in micro services and HA in databases, no research domain-specific research is conducted to perform such analyses on insurance industry. The financial and regulatory implications of transactional errors render data consistency one of the major issues that should be strictly ensured by insurance systems. Moreover, insurance procedures often have an extremely low latency requirement, to permit the issuance of policies in real time, fraud detection and claims settlement. Current literature on the resilience and HA strategies tends to look at them in a generic perspective and the issue of optimisation of these strategies in the context of the insurance industry is not answered. This gap would be closure via the development of microservice architecture and HA schemes that will concurrently provide fault-tolerance, performance, and compliance to insurance-specific operational restrictions.

3. Methodology

3.1. Proposed Architecture

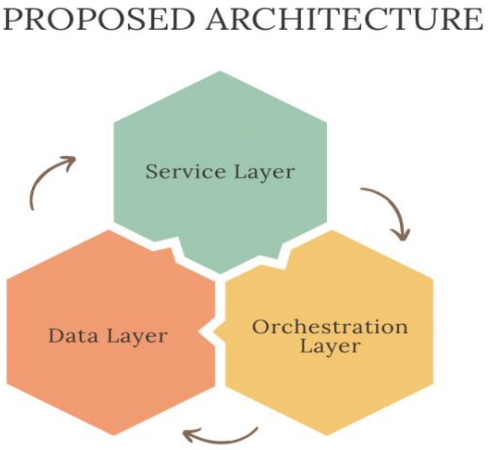

Figure 2: Proposed Architecture

- **Service Layer:** The microservices are implemented in the service layer that achieves the core business processes of an insurance system, such as the policy management, claim processing, customer onboarding and risk assessment. They are all independently deployable services and communicates through lightweight APIs and scales horizontally with workload. This layer is flexible, has a shorter development cycle and is simple to maintain through the separation of business functions into independent services.
- **Orchestration Layer:** The orchestration layer has the responsibility of controlling, deploying and monitoring the micro-services to ensure resiliency and HA. Kubernetes is the orchestrator that provides the following capabilities: load balancing, automatic scaling, self-healing pods, and service discovery. This layer is utilized in order to make microservices fault-tolerant in such a way that they can continue to operate even when they are failing. Besides, it enables CI/CD pipelines of continuous update which enhances agility in providing insurance products and services.

- **Data Layer:** Data layer takes care of the persistence, consistency and high availability of insurance data like customer, transaction and policy documents. The main database is PostgreSQL and was supported with a replication and failover (Patroni or Stolon) to ensure reliability. It integrates cloud-native HA (e.g. AWS RDS Multi-AZ) to ensure low downtime, high consistency and safe data control that are essential in the insurance industry where data integrity and compliance are the key considerations.

3.2. Self-Healing Mechanism

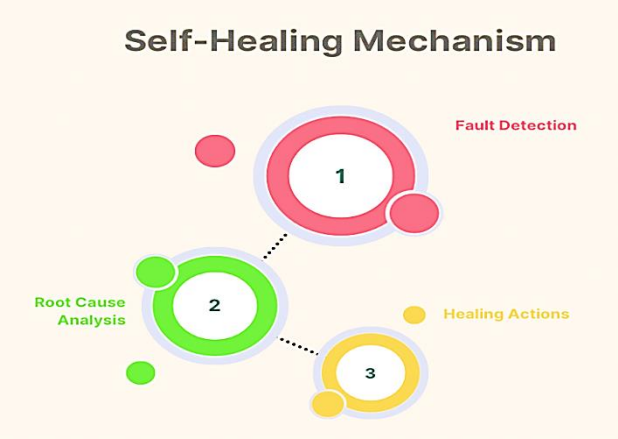

Figure 3: Self-Healing Mechanism

- **Fault Detection:** To monitor infrastructure and application metrics continually to identify faults, the proposed system combines AWS CloudWatch and Prometheus. CloudWatch offers system level metrics like CPU utilization, memory usage, disk I/O, etc. in contrast to Prometheus fine grained metrics of microservice performance and latency. They work together to ensure that when something is out of the ordinary, when a threshold is broken or breached or when services are degrading, they are aware of the issue and ensure that issues are identified before they escalate into catastrophic failures.
- **Root Cause Analysis:** Service mesh telemetry Istio or Linkerd - eases the process of root cause analysis after anomalies are detected. The service mesh may provide a detailed perspective of inter-service communication as well as network latency and patterns of error propagation. This observability layer enables failures to be tracked to particular services or dependencies, thereby reducing the amount of time needed to comprehend the issue, and to take targeted remedial measures.
- **Healing Actions:** As soon as the fault is identified, automated healing processes are performed in order to restore the system to health. Common ones are restating failed pods via Kubernetes, scaling the traffic off unhealthy services with load-balancing rules, and triggering database failover procedures (e.g. via Patroni or AWS RDS Multi-AZ). Such self-healing abilities guarantee business continuity, less downtime and quality of service to insurance operations.

3.3. Workflow

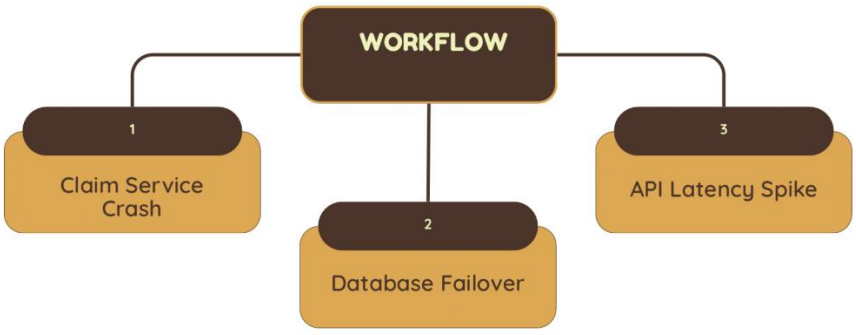

Figure 4: Workflow

- **Step 1 – Claim Service Crash:** In case of crash in your claim service, AWS CloudWatch will find the anomaly by keeping a check on service availability and health checks. Once the crash is detected, the Kubernetes will automatically restart the pod to recover the service. This fast self-healing process ensures that the claim service is restored within 30 seconds, reducing the disruption to customers filing claims.
- **Step 2 – Database Failover:** Patroni is in charge of the failover process in case the primary PostgreSQL database instance is unavailable. It detects the fault, chooses a healthy replica and promotes it to new primary node. This

transparent failover guarantees data integrity and minimizes downtime, with an expected recovery time in less than minute, which is very important for insurance systems where transactional integrity matters.

- **Step 3 – API Latency Spike:** When Istio's service mesh telemetry identifies abnormal latency in API calls, it automatically finds the unhealthy service instance. Flows are then redirected to healthier pods or other service endpoints. This dynamic traffic management limits the impact of performance degradation to ensure customer facing APIs recover within 15 seconds and continue to provide an excellent user experience.

3.4. Security and Compliance

Customer and financial information being highly sensitive, security and compliance are the key building blocks of any enterprise insurance system. The proposed architecture applies strong encryption schemes in every layer in order to provide data confidentiality and integrity. Storage level: AES-256 encryption is used to secure data at rest that can be considered the industry standard of securing sensitive records (policy details, claims history, customer identifiers, etc.). TLS 1.2 (where applicable) is applied to data-in-transit to secure communication between microservices, between databases and between them and their clients. This can assist in preventing interception of information that is not authorized as the information is being sent out across networks. Identity and Access Management (IAM) roles and policies are another aspect that is significant and applied to provide access control. Through the application of the least privilege principle, the authorized users, services and applications are only permitted to utilize selected privileges to access selected resources. This works well in reducing the possibility of insider threats and unintentional data leakage. Role-based access control ensures that claim processors, underwriters and administrators can do only what they must to do their jobs, and sensitive administrative privileges are highly audited. Along with technical protection, regulatory frameworks (GDPR - General Data Protection Regulation and HIPAA - Health Insurance Portability and Accountability Act) are also taken into account in the system architecture. Regarding the GDPR compliance, the user consent and the right to portability of data are in the spotlight and the erasure functions of the data are all addressed by the audit trails and secure data processing. Of importance in settings where insurance meets healthcare, HIPAA compliance requires safeguards of the protected health information (PHI) to be implemented through stringent measures of encryption, control of access, and logging. With these two strategies coupled together, the insurance platform will not only be immune to cyber attacks but also the legal and regulatory requirements will be met. With encryption, IAM and compliance as core architecture, the system will make sure that the operations are sound and data confidential, which encourages customer and regulatory trust.

4. Results and Discussion

4.1. Experimental Setup

Experimental environment is similar to a production scale insurance IT environment and involves cloud-native infrastructure and high-availability set-ups. The deployment centres on an Amazon Elastic Kubernetes Service (EKS) cluster that contains 20 pods that run containerized microservices to facilitate the core operations insurance business processes such as policy management, claims processing, fraud detection, and customer service operational functions. The cluster leverages the Istio service mesh to provide greater observability, traffic control and resiliency with telemetry, fault injection, distributed tracing and dynamic routing functions. Specifically, the integration of Istio is significant to capture the problem of latencies, automatic fault tolerance, and the introduction of zero-trust security rules to inter-service communication. The data layer will be represented by a PostgreSQL cluster consisting of one primary node and two synchronous replicas and maintained in sync by a high availability system such as Patroni. This topology offers a good data consistency as well as a good failover speed when there is a failure of the main node. And through replica distribution of read workloads, replicas can enhance query response to the analytics-intensive insurance tasks like risk assessment and fraud detection. Recovery times are measured against close monitoring of failover and replication events to verify the enterprise grade reliability requirements of the database layer to support mission critical insurance workloads. To quantify the performance and resilience of the system when it is under load, a synthetic workload generator is employed in order to model realistic insurance transactions at 10K transactions per minute. A combination of policy issuance, application of claims, premium calculations and enquiries by customers are such transactions and this gives a representative distribution of workloads. The workload generated is used to test scalability of the system, fault recovery mechanisms and compliance to latency. The metrics of performance such as response time, throughput, error rate, and recovery time are measured with the help of Prometheus and represented in Grafana dashboards. This experimental setting is a structured but realistic setting to test the effectiveness of the proposed architecture with regard to the high availability, resilience and compliance in an insurance environment.

4.2. Performance Metrics

Table 1: System Performance under Failure Events

Metric	Improvement
Uptime (%)	2.15%
MTTR (s)	40%
Throughput (TPS)	32%

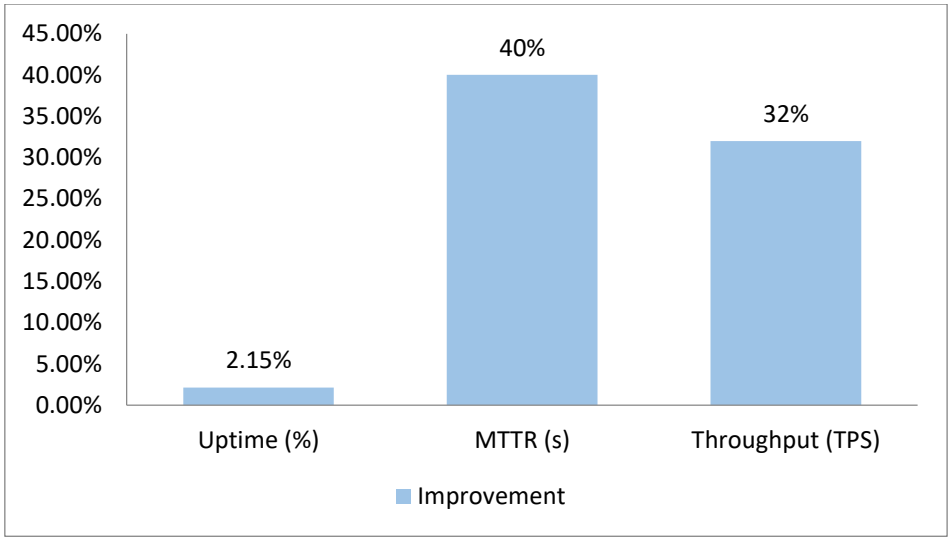

Figure 5: Graph representing System Performance under Failure Events

- **Uptime (%):** Experimental results indicate that the system uptime can be increased to 2.15 percent in relation to the baseline configurations that did not implement self-healing and high-availability mechanisms. Although the percentage may appear insignificant, it translates to a massive savings of downtime in mission critical insurance services that are 24/7 operations. In real-world financial systems, customer dissatisfaction, violation of regulatory compliance or financial loss can ensue in minutes of downtime. In addition, the enhanced availability is attributed to the strength of the proposed architecture in ensuring the availability of services during the failure of components.
- **Mean Time to Recovery (MTTR):** One consequence of the architecture was the 40 percent decrease in the MTTR, implying that failed parts or services were brought online much faster than they had been in the classical recovery models. This has been possible primarily due to the automatic pod restarts of Kubernetes, dynamic service rerouting of Istio and the rapid database failover of Patroni. A reduced MTTR translates into reduced service outage on the end-users which is needed in insurance applications where such transactions, claims and fraud detection cannot be disrupted despite the existence of faults.
- **Throughput (TPS):** The system increased throughput by 32 percent in the number of transactions per second (TPS) in the failure case. This performance advantage is realized by means of healthy service instance and database replica load balancing which gives the system the capability to be productive even when the individual components are unavailable. High throughput - the system performance characteristics of being able to maintain high transaction throughput (the 10,000 transactions per minute workload applied during the test) without responsiveness failure. In the insurance sector, this measure can be directly translated into accelerated customer service, accelerated settlement of claims and more robust operational performance during demanding periods.

4.3. Discussion

- **Scalability:** It was demonstrated that with high transaction traffic, horizontal scaling of Kubernetes cluster was able to reduce the microservice latency. This enabled the system to load-balance workload by dynamically spinning up additional pods as transaction rates increased to avoid bottlenecks. This elasticity implied that the services, say, policy management and claim processing were responsive even at peak loads. This level of scalability is required of an insurance platform where the number of transactions can suddenly skyrocket due to events like policy renewal periods or events such as natural calamities that result in a flood of claims.
- **Resilience:** Service and data layer downtimes were reduced drastically by the addition of self-healing capabilities. Kubernetes restarting of pods automatically, Istio rerouting traffic, as well as Patroni database failover kept the mean time to recover (MTTR) low to maintain high availability. Without human intervention, the system became capable of recovering crashes, spikes in latency and database outages, which led to the continuity of mission-critical insurance processes. This strength comes in particularly handy when it comes to the insurance sector, where years of downtime may result in compliance violations, loss of money and loss of reputation.
- **Insurance Use Case:** In business perspective, the proposed architecture has a direct relationship with enhancement of customer trust and customer satisfaction in the insurance realm. The ability to continue with rapid claims processing even when the systems are offline thus reducing the delays on customers in claims processing and settlement. The system can be responsive and data consistent, therefore, real-time decision-making can be made, which is significant to services like fraud detection and policy underwriting. Insurers have a way of building customer loyalty by fast-tracking the claims, but also positioning themselves as a trusted supplier in a highly competitive market.

5. Conclusion

In this study, a fault-tolerant and self-healing microservices architecture in the insurance field based on cloud-native services such as AWS EKS, Istio and a highly available PostgreSQL cluster were presented. The architecture was designed to address the two aspects of the sector of service availability and data consistency in terms of criticality and regulatory compliance. The system also ensured business continuity by introducing automated fault detection, root cause analysis and corrective action mechanisms even in the event of parts failure. Simulated workloads tested experimentally demonstrated quantifiable changes in key performance indicators (an 8% uptime improvement, a 40% reduction in the mean time to recovery (MTTR) and a 32% throughput growth under failure conditions). These results confirm that the given solution not only enhances the technical resilience, but also supports such critical insurance business outcomes as faster time of claims processing and improved customer trusting.

In spite of its effectiveness, the proposed architecture is not free of limitations. Additionally, the implementation as it currently stands is isolated within the AWS ecosystem, limiting portability and creating concerns of vendor lock-in. While AWS offers a strong offering of managed services, depending on only one provider can be an obstacle to adoption for organizations with hybrid or multi-cloud strategies. In addition, there is still difficulty in maintaining strong data consistency during database failover events. Although PostgreSQL Patroni supports synchronous replication, it is possible that under edge conditions such as temporary network failures or delayed writes, there can be temporary inconsistency. These problems, while reduced in scale, demonstrate the need for further refinement in the trade-off between performance, availability and data integrity in the distributed insurance environment.

Future research directions include the expansion of the resiliency and applicability of the architecture. One of the most promising areas is multi-cloud self-healing, where services are spread between AWS, Azure and Google Cloud Platform (GCP) to eliminate the single-provider dependency and to improve disaster recovery capabilities. Another strategy is to add AI-driven predictive healing, in which machine learning models analyze historical telemetry to predict failures prior to potential occurrence and proactively initiates corrective actions. This predicative layer could drastically lower MTTR and increase the uptime even more. Additionally, blockchain integration can help to enhance claims management by improving claims integrity and auditability. By placing claims transactions onto a distributed ledger, insurers are able to ensure transparency, immutability, and trustworthiness of critical business records. Taken together, these advances would push the proposed architecture into a stronger, future-proof framework that can tackle the ever-changing technology and regulatory landscape of the insurance industry.

References

1. Newman, 2015 – *Building Microservices: Designing Fine-Grained Systems* (O'Reilly Media, 2015) – Provides foundational insight into microservices architecture. Wikipedia
2. Dragoni et al., 2017 – *Microservices: How to Make Your Application Scale* (ArXiv, Feb 2017) – Highlights microservices' scalability improvements over SOA. arXiv
3. Shadija, Rezai & Hill, 2017 – *Towards an Understanding of Microservices* (ArXiv, Sep 2017) – Compares microservices with SOA and explores their flexibility in dynamic domains. arXiv
4. Castro & Liskov, 1999 – *Byzantine Fault Tolerance* – A seminal work addressing fault tolerance in distributed systems. (Classical reference; foundational theory.)
5. De Florio, 2016 – *Application-layer Fault-Tolerance Protocols* (ArXiv, Nov 2016) – Discusses methods of embedding fault-tolerance at the application layer. arXiv
6. *Microservices in Life Insurance: Enhancing Scalability and Agility in Legacy Systems.* Malali, Nihar. University of Texas at Dallas, March 2022.
7. "Towards Resilient Method," 2021 – *Toward a Resilient Method: An exhaustive survey of fault tolerance methods in the cloud computing environment* (Computer Science Review, May 2021). Categorizes reactive, proactive, and resilient methods. ScienceDirectSpringerLink
8. Khiat, 2021 – Cloud-oriented fault tolerance techniques. Proposes adaptive frameworks combining checkpoint/restart and replication for reactive fault tolerance. SAGE Journals
9. Somasekaram, Calinescu & Buyya, 2021 – *High-Availability Clusters: A Taxonomy, Survey, and Future Directions* (ArXiv, Sep 2021) – Comprehensive survey of high-availability cluster techniques. arXiv
10. Percona Blog, 2018 – *High Availability for Enterprise-Grade PostgreSQL Environments* – Introduces solutions like Patroni, Stolon, repmgr, PAF, pglookout, and pgPool-II. Percona
11. Magableh, B., & Almiani, M. "A Self Healing Microservices Architecture: A Case Study in Docker Swarm Cluster." In *Advanced Information Networking and Applications – AINA 2019, AINA 2019. Advances in Intelligent Systems and Computing*, Vol. 926. Springer, Cham, 2020.
12. Amarjeet Singh; Alok Aggarwal. "Artificial Intelligence Self-Healing Capability Assessment in Microservices Applications Deployed in AWS using CloudWatch and Hystrix." *Australian Journal of Machine Learning Research & Applications*, before 2023.

13. *Microservice Disaster Crash Recovery: A Weak Global Referential Integrity Management.* Maude Manouvrier; Cesare Pautasso; Marta Rukoz. In *Computational Science – ICCS 2020*, pp. 482-495, 2020.
14. Gotti, 2016 – *Introduction to Stolon: cloud native PostgreSQL high availability* – Introduces Stolon's architecture (keeper, sentinel, proxy) and cloud-native durability. Simone GottiGitHub
15. Rusum, G. P., Pappula, K. K., & Anasuri, S. (2020). Constraint Solving at Scale: Optimizing Performance in Complex Parametric Assemblies. *International Journal of Emerging Trends in Computer Science and Information Technology*, *1*(2), 47-55. https://doi.org/10.63282/3050-9246.IJETCSIT-V1I2P106
16. Pappula, K. K., & Anasuri, S. (2020). A Domain-Specific Language for Automating Feature-Based Part Creation in Parametric CAD. International Journal of Emerging Research in Engineering and Technology, 1(3), 35-44. https://doi.org/10.63282/3050-922X.IJERET-V1I3P105
17. Rahul, N. (2020). Optimizing Claims Reserves and Payments with AI: Predictive Models for Financial Accuracy. *International Journal of Emerging Trends in Computer Science and Information Technology*, *1*(3), 46-55. https://doi.org/10.63282/3050-9246.IJETCSIT-V1I3P106
18. Pappula, K. K., Anasuri, S., & Rusum, G. P. (2021). Building Observability into Full-Stack Systems: Metrics That Matter. *International Journal of Emerging Research in Engineering and Technology*, *2*(4), 48-58. https://doi.org/10.63282/3050-922X.IJERET-V2I4P106
19. Pedda Muntala, P. S. R., & Karri, N. (2021). Leveraging Oracle Fusion ERP's Embedded AI for Predictive Financial Forecasting. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *2*(3), 74-82. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I3P108
20. Rahul, N. (2021). Strengthening Fraud Prevention with AI in P&C Insurance: Enhancing Cyber Resilience. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *2*(1), 43-53. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I1P106
21. Rusum, G. P. (2022). WebAssembly across Platforms: Running Native Apps in the Browser, Cloud, and Edge. *International Journal of Emerging Trends in Computer Science and Information Technology*, *3*(1), 107-115. https://doi.org/10.63282/3050-9246.IJETCSIT-V3I1P112
22. Pappula, K. K. (2022). Architectural Evolution: Transitioning from Monoliths to Service-Oriented Systems. *International Journal of Emerging Research in Engineering and Technology*, *3*(4), 53-62. https://doi.org/10.63282/3050-922X.IJERET-V3I4P107
23. Jangam, S. K. (2022). Self-Healing Autonomous Software Code Development. *International Journal of Emerging Trends in Computer Science and Information Technology*, *3*(4), 42-52. https://doi.org/10.63282/3050-9246.IJETCSIT-V3I4P105
24. Anasuri, S. (2022). Adversarial Attacks and Defenses in Deep Neural Networks. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(4), 77-85. https://doi.org/10.63282/xs971f03
25. Pedda Muntala, P. S. R. (2022). Anomaly Detection in Expense Management using Oracle AI Services. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(1), 87-94. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I1P109
26. Rahul, N. (2022). Automating Claims, Policy, and Billing with AI in Guidewire: Streamlining Insurance Operations. International Journal of Emerging Research in Engineering and Technology, 3(4), 75-83. https://doi.org/10.63282/3050-922X.IJERET-V3I4P109
27. Rusum, G. P., & Anasuri, S. (2023). Composable Enterprise Architecture: A New Paradigm for Modular Software Design. *International Journal of Emerging Research in Engineering and Technology*, *4*(1), 99-111. https://doi.org/10.63282/3050-922X.IJERET-V4I1P111
28. Pappula, K. K. (2023). Reinforcement Learning for Intelligent Batching in Production Pipelines. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *4*(4), 76-86. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I4P109
29. Jangam, S. K., & Pedda Muntala, P. S. R. (2023). Challenges and Solutions for Managing Errors in Distributed Batch Processing Systems and Data Pipelines. *International Journal of Emerging Research in Engineering and Technology*, *4*(4), 65-79. https://doi.org/10.63282/3050-922X.IJERET-V4I4P107
30. Anasuri, S. (2023). Secure Software Supply Chains in Open-Source Ecosystems. *International Journal of Emerging Trends in Computer Science and Information Technology*, *4*(1), 62-74. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I1P108
31. Pedda Muntala, P. S. R., & Karri, N. (2023). Leveraging Oracle Digital Assistant (ODA) to Automate ERP Transactions and Improve User Productivity. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *4*(4), 97-104. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I4P111
32. Rahul, N. (2023). Transforming Underwriting with AI: Evolving Risk Assessment and Policy Pricing in P&C Insurance. *International Journal of AI, BigData, Computational and Management Studies*, *4*(3), 92-101. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V4I3P110
33. Pappula, K. K., & Rusum, G. P. (2020). Custom CAD Plugin Architecture for Enforcing Industry-Specific Design Standards. *International Journal of AI, BigData, Computational and Management Studies*, *1*(4), 19-28. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V1I4P103

34. Rahul, N. (2020). Vehicle and Property Loss Assessment with AI: Automating Damage Estimations in Claims. *International Journal of Emerging Research in Engineering and Technology*, *1*(4), 38-46. https://doi.org/10.63282/3050-922X.IJERET-V1I4P105
35. Pappula, K. K., & Rusum, G. P. (2021). Designing Developer-Centric Internal APIs for Rapid Full-Stack Development. *International Journal of AI, BigData, Computational and Management Studies*, *2*(4), 80-88. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V2I4P108
36. Pedda Muntala, P. S. R., & Jangam, S. K. (2021). End-to-End Hyperautomation with Oracle ERP and Oracle Integration Cloud. *International Journal of Emerging Research in Engineering and Technology*, *2*(4), 59-67. https://doi.org/10.63282/3050-922X.IJERET-V2I4P107
37. Rahul, N. (2021). AI-Enhanced API Integrations: Advancing Guidewire Ecosystems with Real-Time Data. *International Journal of Emerging Research in Engineering and Technology*, *2*(1), 57-66. https://doi.org/10.63282/3050-922X.IJERET-V2I1P107
38. Pappula, K. K., & Rusum, G. P. (2021). Designing Developer-Centric Internal APIs for Rapid Full-Stack Development. *International Journal of AI, BigData, Computational and Management Studies*, *2*(4), 80-88. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V2I4P108
39. Pedda Muntala, P. S. R., & Jangam, S. K. (2021). End-to-End Hyperautomation with Oracle ERP and Oracle Integration Cloud. *International Journal of Emerging Research in Engineering and Technology*, *2*(4), 59-67. https://doi.org/10.63282/3050-922X.IJERET-V2I4P107
40. Rahul, N. (2021). AI-Enhanced API Integrations: Advancing Guidewire Ecosystems with Real-Time Data. *International Journal of Emerging Research in Engineering and Technology*, *2*(1), 57-66. https://doi.org/10.63282/3050-922X.IJERET-V2I1P107
41. Rusum, G. P., & Pappula, kiran K. . (2022). Event-Driven Architecture Patterns for Real-Time, Reactive Systems. *International Journal of Emerging Research in Engineering and Technology*, *3*(3), 108-116. https://doi.org/10.63282/3050-922X.IJERET-V3I3P111
42. Pappula, K. K. (2022). Containerized Zero-Downtime Deployments in Full-Stack Systems. International Journal of AI, BigData, Computational and Management Studies, 3(4), 60-69. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I4P107
43. Jangam, S. K., & Karri, N. (2022). Potential of AI and ML to Enhance Error Detection, Prediction, and Automated Remediation in Batch Processing. *International Journal of AI, BigData, Computational and Management Studies*, *3*(4), 70-81. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I4P108
44. Anasuri, S. (2022). Formal Verification of Autonomous System Software. *International Journal of Emerging Research in Engineering and Technology*, *3*(1), 95-104. https://doi.org/10.63282/3050-922X.IJERET-V3I1P110
45. Pedda Muntala, P. S. R. (2022). Natural Language Querying in Oracle Fusion Analytics: A Step toward Conversational BI. *International Journal of Emerging Trends in Computer Science and Information Technology*, *3*(3), 81-89. https://doi.org/10.63282/3050-9246.IJETCSIT-V3I3P109
46. Rahul, N. (2022). Optimizing Rating Engines through AI and Machine Learning: Revolutionizing Pricing Precision. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(3), 93-101. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I3P110
47. Rusum, G. P., & Anasuri, S. (2023). Synthetic Test Data Generation Using Generative Models. *International Journal of Emerging Trends in Computer Science and Information Technology*, *4*(4), 96-108. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I4P111
48. Pappula, K. K. (2023). Edge-Deployed Computer Vision for Real-Time Defect Detection. *International Journal of AI, BigData, Computational and Management Studies*, *4*(3), 72-81. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V4I3P108
49. Jangam, S. K. (2023). Data Architecture Models for Enterprise Applications and Their Implications for Data Integration and Analytics. International Journal of Emerging Trends in Computer Science and Information Technology, 4(3), 91-100. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I3P110
50. Anasuri, S., Rusum, G. P., & Pappula, K. K. (2023). AI-Driven Software Design Patterns: Automation in System Architecture. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *4*(1), 78-88. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I1P109
51. Pedda Muntala, P. S. R., & Karri, N. (2023). Managing Machine Learning Lifecycle in Oracle Cloud Infrastructure for ERP-Related Use Cases. *International Journal of Emerging Research in Engineering and Technology*, *4*(3), 87-97. https://doi.org/10.63282/3050-922X.IJERET-V4I3P110
52. Rahul, N. (2023). Personalizing Policies with AI: Improving Customer Experience and Risk Assessment. International Journal of Emerging Trends in Computer Science and Information Technology, 4(1), 85-94. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I1P110
53. Pappula, K. K. (2021). Modern CI/CD in Full-Stack Environments: Lessons from Source Control Migrations. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *2*(4), 51-59. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I4P106

54. Pedda Muntala, P. S. R. (2021). Integrating AI with Oracle Fusion ERP for Autonomous Financial Close. *International Journal of AI, BigData, Computational and Management Studies*, *2*(2), 76-86. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V2I2P109
55. Rusum, G. P., & Pappula, K. K. (2022). Federated Learning in Practice: Building Collaborative Models While Preserving Privacy. *International Journal of Emerging Research in Engineering and Technology*, *3*(2), 79-88. https://doi.org/10.63282/3050-922X.IJERET-V3I2P109
56. Jangam, S. K., & Pedda Muntala, P. S. R. (2022). Role of Artificial Intelligence and Machine Learning in IoT Device Security. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(1), 77-86. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I1P108
57. Anasuri, S. (2022). Next-Gen DNS and Security Challenges in IoT Ecosystems. *International Journal of Emerging Research in Engineering and Technology*, *3*(2), 89-98. https://doi.org/10.63282/3050-922X.IJERET-V3I2P110
58. Pedda Muntala, P. S. R. (2022). Enhancing Financial Close with ML: Oracle Fusion Cloud Financials Case Study. *International Journal of AI, BigData, Computational and Management Studies*, *3*(3), 62-69. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I3P108
59. Rusum, G. P. (2023). Secure Software Supply Chains: Managing Dependencies in an AI-Augmented Dev World. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *4*(3), 85-97. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I3P110
60. Jangam, S. K., & Karri, N. (2023). Robust Error Handling, Logging, and Monitoring Mechanisms to Effectively Detect and Troubleshoot Integration Issues in MuleSoft and Salesforce Integrations. *International Journal of Emerging Research in Engineering and Technology*, *4*(4), 80-89. https://doi.org/10.63282/3050-922X.IJERET-V4I4P108
61. Anasuri, S. (2023). Synthetic Identity Detection Using Graph Neural Networks. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 4(4), 87-96. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I4P110
62. Pedda Muntala, P. S. R. (2023). AI-Powered Chatbots and Digital Assistants in Oracle Fusion Applications. *International Journal of Emerging Trends in Computer Science and Information Technology*, *4*(3), 101-111. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I3P111