

다중 영역 Kubernetes 환경에서 자체 복구 및 확장을 위한 자율형 PostgreSQL 오퍼레이터

타히라 아민*, 무하마드 아시프*, 우마르 마흐무드*, 송 왕 철°

Autonomous PostgreSQL Operator for Self-healing and Scaling in Multi-Domain Kubernetes Environments

Tahira Amin*, Muhammad Asif*, Umar Mahmood*, Wang-Cheol Song°

ABSTRACT

The increasing adoption of Kubernetes for cloud-native applications has intensified the need for reliable and autonomous management of stateful database services, particularly in multi-zone environments where failures and topology changes are expected. Although Kubernetes Operators offer declarative automation for managing complex workloads, their effectiveness in handling production-grade database lifecycles remains insufficiently explored. This paper presents an experimental evaluation of an autonomous PostgreSQL Operator deployed in a multi-zone Kubernetes cluster, focusing on its ability to manage key operational scenarios without manual intervention. The study investigates horizontal scaling, self-healing and automated failover, configuration-triggered zone relocation, and rolling upgrades through controlled configuration updates and fault injection. Operator behavior is analyzed using Kubernetes events, pod lifecycle transitions, and cluster state observations. Results show that scaling operations are executed in a consistency-preserving manner through synchronized replica initialization, primary failures are resolved autonomously via safe replica promotion, topology changes correctly trigger zone-aware re-orchestration, and rolling upgrades are performed sequentially while maintaining service availability. Across all scenarios, the Operator relies solely on Kubernetes-native declarative control loops to reconcile observed and desired system state. These findings demonstrate that PostgreSQL Operators can function as effective autonomous control systems for managing stateful databases in multi-zone Kubernetes environments, reducing operational complexity and supporting resilient, production-grade cloud-native deployments.

Key Words : Kubernetes Operators, Autonomous Database Management, Multi-Zone Kubernetes, PostgreSQL, Cloud-Native.

※본 연구는 한국연구재단 논문연구과제(95-0100-23-04-5) 지원 및 한국대학교 논문연구소 관리로 수행되었습니다.

◆ First Author : Jeju National University Department of Electronic Engineering, tahiraamin95@stu.jejunu.ac.kr

° Corresponding Author : Jeju National University Department of Computer Engineering, philo@jejunu.ac.kr

* Jeju National University Department of Computer and Electronic Engineering, asif07@office.jejunu.ac.kr,
umarmahmood637@gmail.com

논문번호 : KNOM2025-02-013, Received November 15, 2025; Revised December 2, 2025; Accepted December 16, 2025

I. Introduction

Modern cloud-native applications increasingly depend on stateful services to support reliable data management, transactional integrity, and real-time analytics in distributed environments. Despite the widespread adoption of microservices and Kubernetes-based orchestration, relational databases remain fundamental to enterprise and scientific workloads. Operating such databases in Kubernetes, however, introduces significant challenges related to availability, fault tolerance, and elastic scalability, particularly when deployments span geographically or topologically separated availability zones. In multi-zone Kubernetes environments, failures are expected operational conditions rather than rare events, ranging from individual pod or node failures to partial or complete zone outages. Manual handling of such events is slow, error-prone, and incompatible with the scale and agility required by modern cloud platforms.

Numerous approaches have attempted to address these challenges. Traditional high-availability database solutions often rely on external clustering software, static replication configurations, or infrastructure assumptions that conflict with Kubernetes' declarative and dynamic design principles. Early Kubernetes deployments frequently treated databases as fixed entities, relying on manual failover procedures and tight coupling to specific nodes. Although Kubernetes Operators introduced a more structured automation model—by encapsulating operational knowledge into controllers that continuously reconcile a desired system state with the observed runtime state—many existing solutions still focus primarily on basic lifecycle tasks such as deployment and backup. As a result, support for autonomous self-healing, zone-aware failover, and elastic scaling under realistic failure scenarios remains limited, and significant manual oversight is often required in multi-zone deployments with complex failure modes.

This gap motivates the central research question of this study: can a PostgreSQL Operator deployed in a multi-zone Kubernetes environment autonomously deliver reliable self-healing and scalable behavior suitable for production-grade stateful workloads? Beyond basic feasibility, this question also concerns whether such automation is observable, reproducible, and robust under controlled fault, scaling, relocation, and upgrade scenarios. This paper advances the hypothesis that a well-designed PostgreSQL Operator—built on Kubernetes-native primitives, declarative control loops, and zone-aware scheduling mechanisms—can function as an autonomous control system for managing stateful databases in dynamic, failure-prone environments.

In this context, a declarative control loop refers to the continuous process by which the Operator monitors cluster state through Kubernetes events and resource status, and automatically triggers corrective actions when deviations from the desired configuration are detected. Zone-aware scheduling enables database instances to be placed across distinct failure domains, such as availability zones, to improve resilience against localized disruptions. Together, these mechanisms form the foundation of the autonomous behaviors evaluated in this study.

The main contributions of this work include: (i) deployment of a PostgreSQL Operator - managed cluster in a multi-node, multi-zone Kubernetes environment; (ii) experimental evaluation of autonomous self-healing and failover behavior; (iii) analysis of horizontal scaling operations; (iv) investigation of configuration-triggered pod relocation across zones; and (v) examination of rolling update behavior during PostgreSQL upgrades. Rather than proposing a new database operator or control algorithm, this work focuses on a systematic, scenario-driven evaluation of autonomous operator behavior in a multi-zone Kubernetes environment. The contribution lies in the structured validation of declarative control loops under realistic operational conditions—such as failure recovery, topology-aware scheduling,

and rolling lifecycle events—which are often underexplored in existing operator evaluations.

The remainder of the paper is organized as follows. Section 2 reviews related work, Section 3 presents the methodology and system architecture, Section 4 reports experimental results, and Section 5 concludes the paper.

II. Related Work

The orchestration of stateful services in Kubernetes has emerged as a central research challenge as cloud-native platforms evolve toward production-grade data management. Although Kubernetes was initially designed for stateless workloads, recent studies show that managing databases introduces additional complexity related to scheduling, availability, fault tolerance, and lifecycle automation. Surveys and taxonomies of Kubernetes scheduling mechanisms highlight limitations of native schedulers in handling stateful workloads, topology awareness, and multi-objective constraints [1], [13].

Broader cloud-native research emphasizes that stateful services require orchestration mechanisms beyond basic container placement and restart semantics. Service-centric surveys underline the growing need for intelligent lifecycle management, resilience, and automation in distributed systems,

motivating the adoption of higher-level abstractions such as Kubernetes Operators and custom controllers[2]. At the same time, reliability-focused studies reveal that declarative infrastructure and control-loop-based management are prone to misconfigurations and subtle bugs, which can propagate into large-scale failures if not carefully evaluated [4], [14],[15]. Self-healing and disaster recovery have been explored through automated recovery frameworks and proactive lifecycle management strategies for stateful microservices.

These works highlight the importance of coordinated recovery and replica management, particularly in multi-cluster and failure-prone environments [6], [7]. Scaling of stateful workloads has also attracted attention, with experimental studies showing that naive horizontal scaling fails to account for replication overhead and consistency constraints [11]. Subsequent research proposes hybrid and rule-based scaling approaches to improve resilience and burst tolerance under dynamic workloads [15], [9].

Topology-aware orchestration, migration, and relocation represent another important research direction. Scheduling studies propose SLA-aware and multi-objective placement strategies, while migration frameworks enable runtime relocation of workloads across nodes or clusters using

Table 1. Comparative analysis of representative works, methodologies, and limitations in Kubernetes-based orchestration of stateful services

Ref.	Research Focus	Methodology	Key Limitation
(Carrión, 2022)	Kubernetes scheduling limitations	Survey and taxonomy	No stateful database-specific evaluation
(Deng et al., 2024)	Cloud-native service orchestration	Comprehensive survey	Broad scope, non-operator-specific
(Drosos et al., 2024)	Reliability of declarative infrastructure	Empirical fault analysis	Does not address runtime recovery
(Kim et al., 2024)	Automated disaster recovery in Kubernetes	System design and experimental evaluation	Focused on recovery only
(Meliani et al., 2025)	Proactive resiliency for stateful microservices	Experimental lifecycle management	Multi-cluster focus, not zone-specific
(Tran & Kim, 2024)	Hybrid auto-scaling for cloud/edge systems	Experimental evaluation	Serverless-oriented, not database-focused
(Dinh-Tuan & Jiang, 2025)	Stateful microservice migration	Checkpoint-based experimental study	Migration-focused, not full lifecycle

checkpointing and dynamic scheduling techniques [5], [10], [3]. Finally, rolling updates and zero-downtime deployment strategies have been studied in the context of cloud applications, highlighting trade-offs between availability and operational complexity and motivating controlled, sequential update mechanisms [12], [8]. Table 1 summarizes representative approaches to Kubernetes-based orchestration of stateful services, highlighting differences in methodology and evaluation scope.

III. Methodology and System Architecture

This section presents the methodology and system architecture of the autonomous PostgreSQL Operator evaluated in this study. The system is

designed as a Kubernetes-native control framework for managing PostgreSQL clusters deployed across multiple availability zones. The architecture emphasizes autonomous operation, observability, and reproducible evaluation under failure, scaling, and topology change scenarios. Figure 1 illustrates the overall architecture and interactions among system components.

3.1 Architectural Overview

The proposed architecture follows a control-loop-driven design, a common paradigm in Kubernetes-based systems, in which a PostgreSQL Operator continuously reconciles the desired cluster configuration with the observed runtime state of the system. In this context, the desired state is specified declaratively through custom resources, while the observed state is derived

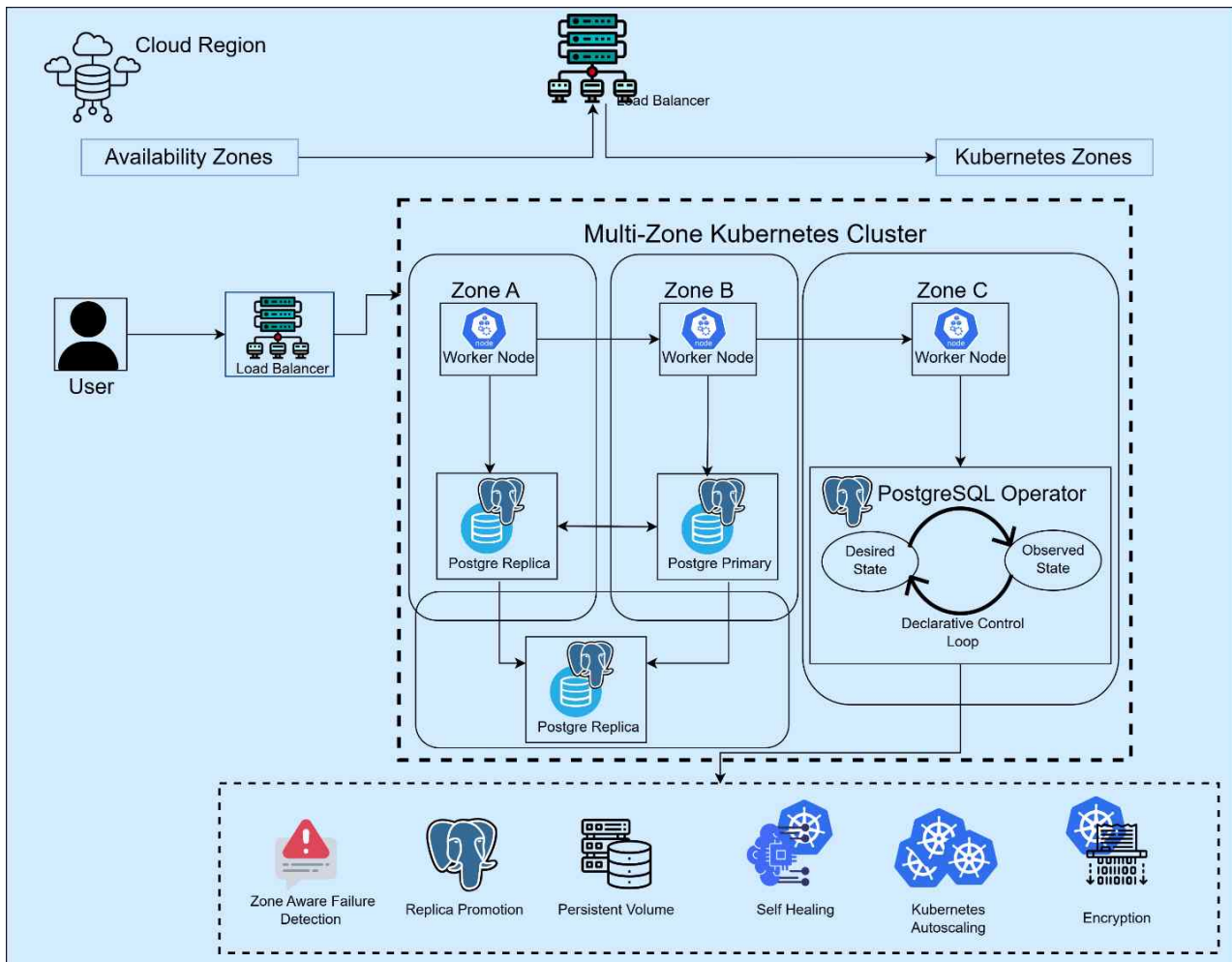


Fig. 1. Autonomous PostgreSQL Operator architecture deployed in a multi-zone Kubernetes environment

from Kubernetes events, pod status, and database health information. Any deviation between the two triggers corrective actions initiated by the Operator.

The system is deployed on a multi-zone Kubernetes cluster, where worker nodes are distributed across distinct zones to represent realistic cloud failure domains. Client access to the PostgreSQL service is provided through Kubernetes Services and an external load balancer, which transparently routes requests to the active primary instance and maintains connectivity during failover events.

At a high level, the architecture consists of four interacting layers: (i) the Kubernetes infrastructure layer, which provides scheduling and resource management; (ii) the PostgreSQL cluster deployment layer, which defines database instances and storage; (iii) the Operator control layer, which encodes autonomous management logic; and (iv) the observability and evaluation layer, which collects runtime information for analysis. These layers communicate exclusively through Kubernetes-native APIs, events, and declarative specifications, forming a closed-loop system capable of responding autonomously to failures, scaling requests, and topology changes. PostgreSQL instances are deployed using StatefulSets in a primary-replica configuration, with durable storage provided by Persistent Volumes to preserve data across restarts and role transitions.

3.2 Multi-Zone Infrastructure and Cluster Deployment

The Kubernetes cluster is configured with explicit zone labels to enable zone-aware scheduling, which allows the placement of database instances across distinct failure domains. This approach improves resilience by reducing the impact of localized node or zone failures. Persistent Volumes are provisioned using zone-scoped storage classes to ensure data locality and durability while avoiding unintended cross-zone dependencies.

Cluster properties, including replica count, resource allocation, and zone constraints are defined declaratively through custom resource definitions. These specifications serve as the authoritative input to the Operator's reconciliation logic, allowing configuration changes to be applied consistently and reproducibly without manual intervention.

3.3 PostgreSQL Operator Control Loop

The PostgreSQL Operator implements a continuous reconciliation loop that monitors pod status, replication health, storage bindings, and Kubernetes events. Failure detection is zone-aware, enabling the Operator to distinguish between localized pod or node failures and broader zone-level disruptions. When deviations from the desired state are detected, the Operator autonomously initiates corrective actions such as replica promotion, pod re-creation, or zone-aware redistribution of database instances.

Scaling operations are handled through controlled replica provisioning and deprovisioning, with synchronization and readiness checks enforced to preserve data consistency. This control-loop-based approach enables the Operator to manage complex stateful behaviors without relying on external orchestration logic or manual oversight.

3.4 Observability and Methodological Workflow

Observability components collect Kubernetes events, Operator logs, and PostgreSQL metrics to support evaluation of system behavior. These data sources provide visibility into recovery time, failover latency, replica readiness, and scaling stabilization behavior, while remaining strictly separated from the control logic to avoid influencing operational decisions.

The evaluation workflow begins with declarative cluster deployment and stabilization, followed by controlled fault injection and scaling events. System behavior is monitored until the cluster returns to a stable state, enabling repeatable and

objective assessment of autonomous self-healing and scaling behavior in a multi-zone Kubernetes environment.

3.5 Observability and Evaluation

The architecture integrates observability mechanisms to collect runtime information required for experimental analysis. Kubernetes events, Operator logs, and database-level metrics are used to evaluate recovery behavior, role transitions, and stabilization times. By maintaining a clear separation between observability and control, the experimental results reflect the inherent behavior of the Operator rather than external instrumentation effects, supporting reproducible evaluation.

3.6 Methodological Workflow

The methodological workflow begins with the declarative deployment of a PostgreSQL cluster across multiple zones, followed by stabilization and baseline metric collection. Controlled events including pod failures, primary termination, and replica scaling are then introduced to trigger autonomous responses by the Operator. System behavior is continuously observed until the cluster returns to a stable state, enabling objective assessment of self-healing effectiveness, scaling behavior, and topology-aware re-orchestration.

IV. Experimental Results

This section evaluates the autonomous behavior of the PostgreSQL Operator under controlled experimental scenarios, focusing on horizontal scaling, self-healing and failover, configuration-triggered zone relocation, and rolling upgrades. Experiments were conducted in a reproducible multi-node Kubernetes environment using declarative configuration and Kubernetes-native control mechanisms. The goal is not performance benchmarking, but systematic observation of operator-driven lifecycle management under realistic operational events. In addition to qualitative verification of state

transitions, quantitative timing indicators were extracted from Kubernetes pod lifecycle timestamps and Operator event logs to summarize the responsiveness of autonomous operations.

4.1 Experimental Environment and Baseline

Experiments were performed on a Kubernetes cluster created using `kind`, consisting of one control-plane node and two worker nodes. To emulate a multi-zone deployment, worker nodes were labeled as belonging to separate zones, enabling evaluation of zone-aware scheduling without geographically distributed infrastructure. The CloudNativePG PostgreSQL Operator was deployed to manage a three-instance PostgreSQL cluster defined declaratively through a custom resource. Each instance was provisioned with persistent storage and node affinity constraints. After deployment, the cluster stabilized with all pods in a running and ready state, forming the baseline for subsequent experiments.

4.2 Horizontal Scaling

Horizontal scaling was evaluated by increasing the desired replica count from three to five through a declarative configuration update. Upon applying the updated cluster specification, the PostgreSQL Operator automatically initiated the scaling process without manual intervention. The operator first created temporary join pods to synchronize data between the existing cluster members and the newly provisioned replicas. During this phase, newly created pods transitioned through initialization and synchronization states before becoming active members of the cluster.

After the data synchronization phase completed, all PostgreSQL pods reached the Running and Ready state, resulting in a stable five-instance cluster configuration. Based on Kubernetes pod lifecycle timestamps and event ordering, the data synchronization phase required approximately 25 - 30 seconds, while the full scaling stabilization time measured from configuration update to all replicas becoming ready was approximately 40 -

45 seconds. This behavior demonstrates that the operator performs controlled and consistency-preserving scaling by enforcing synchronization and readiness checks before exposing new replicas to the cluster.

Table 2 summarizes the observed outcomes and stabilization time of the horizontal scaling experiment, as derived from Kubernetes pod lifecycle events and Operator logs.

Table 2. Observed outcomes for the horizontal scaling experiment

Scaling Attribute	Observed Outcome
Initial replica count	3
Target replica count	5
Scaling trigger	Declarative configuration update
Temporary join pods	Created for data synchronization
Data synchronization duration	~25 - 30 s
Full stabilization time	~40 - 45 s
Manual intervention required	No

4.3 Self-Healing and Failover

Self-healing behavior was examined by deliberately deleting the active primary PostgreSQL pod identified through cluster status inspection. Upon failure injection, the PostgreSQL Operator detected the primary unavailability and autonomously promoted a healthy replica to the primary role while recreating the failed pod, without requiring manual intervention.

Based on Kubernetes event ordering and pod lifecycle timestamps, failure detection occurred within approximately 5 - 7 seconds, and replica promotion was completed within 10 - 12 seconds after detection. Full cluster recovery, including recreation of the failed pod and readiness verification, was achieved within approximately 30 - 35 seconds. The cluster returned to a healthy state with all pods running and ready, confirming the Operator's ability to maintain service availability through autonomous failover.

Table 3 summarizes the observed outcomes and recovery characteristics of the self-healing and failover experiment.

Table 3. Observed outcomes for the self-healing and failover experiment

Failover Attribute	Observed Outcome
Failure type	Primary pod deletion
Failure detection time	~5 - 7 s
Replica promotion time	~10 - 12 s
Failed pod re-creation	Automatically initiated
Full recovery time	~30 - 35 s
Manual intervention required	No
Service availability	Maintained during failover

4.4 Zone Relocation

To evaluate topology-aware relocation, node affinity constraints were modified to target a different availability zone through a declarative configuration update. After applying the updated configuration, the PostgreSQL Operator autonomously terminated existing pods and re-created PostgreSQL instances on nodes matching the new zone label. During this re-orchestration process, temporary initialization and join pods were created to ensure correct data synchronization and role assignment.

Based on Kubernetes event ordering and pod lifecycle timestamps, pod re-creation and relocation required approximately 20 - 25 seconds, while full cluster stabilization in the target zone—measured from configuration update to all pods reaching a running and ready state—was achieved within approximately 35 - 40 seconds. These observations demonstrate correct enforcement of declarative topology constraints and confirm the Operator's ability to perform autonomous, zone-aware re-orchestration without manual intervention.

Table 4 summarizes the observed outcomes and stabilization characteristics of the zone relocation experiment.

Table 4. Observed outcomes for the zone relocation experiment

Relocation Attribute	Observed Outcome
Relocation trigger	Node affinity configuration update
Source zone	Original deployment zone
Target zone	Updated affinity zone
Pod re-creation initiated	Yes
Temporary initialization/join pods	Observed during relocation
Pod relocation duration	~20 - 25 s
Full stabilization time	~35 - 40 s
Manual intervention required	No

4.5 Rolling Upgrades

Rolling upgrade behavior was evaluated by updating the PostgreSQL container image from version 15 to version 16 through a declarative configuration change. Upon applying the updated specification, the PostgreSQL Operator autonomously initiated a rolling upgrade procedure, sequentially restarting individual pods. Each PostgreSQL instance was upgraded only after the previously updated pod reached a running and ready state, ensuring that service availability was maintained throughout the process. The upgrade completed successfully with all PostgreSQL pods running the updated version, which was verified through in-database version checks. No manual intervention was required during the upgrade, and no service interruption was observed. These results confirm that the Operator correctly enforces ordered pod restarts and readiness checks to support safe and autonomous rolling upgrades.

Table 5 summarizes the observed outcomes of the rolling upgrade experiment.

Table 5. Observed outcomes for the rolling upgrade experiment

Upgrade Attribute	Observed Outcome
Upgrade trigger	PostgreSQL image update (v15 → v16)
Upgrade strategy	Sequential pod restarts
Readiness enforcement	Yes (one pod at a time)
Service availability	Maintained throughout upgrade
Manual intervention required	No
Upgrade verification	In-database version check
Final cluster state	All pods running PostgreSQL v16

Across all experiments, the PostgreSQL Operator

exhibited consistent autonomous behavior aligned with its declarative control model. Scaling, failover, relocation, and upgrades were handled without manual intervention and within bounded stabilization times, validating the Operator's capability to manage stateful database lifecycles in a multi-zone Kubernetes environment using Kubernetes-native mechanisms.

Table 6 summarizes the observed stabilization times for autonomous scaling, failover, relocation, and upgrade operations extracted from Kubernetes event logs and pod lifecycle transitions.

Table 6. Summary of observed stabilization times for autonomous PostgreSQL Operator operations

Scenario	Trigger Event	Observed Stabilization Time
Horizontal scaling	Replica count update (3 → 5)	~40 - 45 s
Self-healing / failover	Primary pod deletion	~30 - 35 s
Zone relocation	Node affinity update	~35 - 40 s
Rolling upgrade	PostgreSQL image update (v15 → v16)	Sequential, no service interruption

V. Conclusion and Future Work

This paper presented a systematic evaluation of an autonomous PostgreSQL Operator deployed in a multi zone Kubernetes environment, with the goal of understanding how declarative, Kubernetes native control mechanisms can be used to manage stateful database lifecycles. A range of operational scenarios was examined, including horizontal scaling, self healing and automated failover, configuration driven zone relocation, and rolling upgrades, in order to observe how the Operator responds to dynamic conditions without manual intervention.

The experimental results show that scaling operations progress in a controlled and consistency preserving manner, primary failures are handled autonomously through safe replica promotion, and topology changes correctly trigger zone aware relocation of database instances. In addition, rolling upgrades were carried out sequentially while maintaining service availability, indicating

that routine version transitions can be supported without requiring full service disruption.

Overall, these findings suggest that PostgreSQL Operators can function as effective autonomous control systems for managing stateful databases in Kubernetes, particularly in multi zone environments where reconciliation logic, topology awareness, and storage integration must work together correctly. The experiments were conducted using a kind based Kubernetes cluster with a limited number of nodes, which allowed for controlled execution and clear observation of operator behavior. While this environment is sufficient for validating reconciliation logic and operational correctness, it does not fully reflect the scale, infrastructure diversity, or network characteristics typically found in production cloud deployments.

When deployed in larger clusters or managed Kubernetes platforms such as GKE or EKS, additional factors including inter zone network latency, cloud managed storage behavior, autoscaling interactions, and higher replica counts may influence failover and recovery characteristics. Even so, because the PostgreSQL Operator relies on Kubernetes native declarative reconciliation mechanisms, the qualitative behaviors observed in this study are expected to carry over to real world environments. Future work will focus on evaluating performance under load, comparing different operator implementations, and extending the analysis to larger or geographically distributed deployments.

References

- [1]. Carrión, C. (2022). Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges. *ACM Comput. Surv.*, 55(7), 138:1-138:37. <https://doi.org/10.1145/3539606>
- [2]. Deng, S., Zhao, H., Huang, B., Zhang, C., Chen, F., Deng, Y., Yin, J., Dustdar, S., & Zomaya, A. Y. (2024). Cloud-Native Computing: A Survey From the Perspective of Services. *Proceedings of the IEEE*, 112(1), 12 - 4 6 . <https://doi.org/10.1109/JPROC.2024.3353855>
- [3]. Dinh-Tuan, H., & Jiang, J. (2025). Optimizing Stateful Microservice Migration in Kubernetes with MS2M and Forensic Checkpointing. 2025 28th Conference on Innovation in Clouds, Internet and Networks (ICIN), 83 - 90. <https://doi.org/10.1109/ICIN64016.2025.10942720>
- [4]. Drosos, G.-P., Sotiropoulos, T., Alexopoulos, G., Mitropoulos, D., & Su, Z. (2024). When Your Infrastructure Is a Buggy Program: Understanding Faults in Infrastructure as Code Ecosystems. *Reproduction Package for Article: "When Your Infrastructure Is a Buggy Program: Understanding Faults in Infrastructure as Code Ecosystems"*, 8(OOPSLA2), 3 5 9 : 2 4 9 0 - 3 5 9 : 2 5 2 0 . <https://doi.org/10.1145/3689799>
- [5]. Farid, M., Lim, H. S., Lee, C. P., Zarakovitis, C. C., & Chien, S. F. (2025). Optimizing Kubernetes with Multi-Objective Scheduling Algorithms: A 5G Perspective. *Computers*, 14(9), 390. <https://doi.org/10.3390/computers14090390>
- [6]. Kim, J.-B., Choi, J.-B., & Jung, E.-S. (2024). Design and Implementation of an Automated Disaster-Recovery System for a Kubernetes Cluster Using LSTM. *Applied Sciences*, 14(9), 3914. <https://doi.org/10.3390/app14093914>
- [7]. Meliani, A. E., Mekki, M., & Ksentini, A. (2025). Resiliency focused proactive lifecycle management for stateful microservices in multi-cluster containerized environments. *Computer Communications*, 236, 108111. <https://doi.org/10.1016/j.comcom.2025.108111>
- [7]. Mondal, S. K., Zheng, Z., & Cheng, Y. (2024). On the Optimization of Kubernetes toward the Enhancement of Cloud Computing. *Mathematics*, 12(16), 2476. <https://doi.org/10.3390/math12162476>
- [8]. Pakrijauskas, K., & Mažeika, D. (2025). A Rule-Based Method for Enhancing Burst Tolerance in Stateful Microservices. *Electronics*, 14(14), 2752.

- <https://doi.org/10.3390/electronics14142752> and Analysis, 1746 – 1758.
<https://doi.org/10.1145/3650212.36>
- [9]. Pashaeehir, A., Shariati, S., Shafaghi, S., Moghimi, M., & Momtazpour, M. (2025). KubeDSM: A Kubernetes-based Dynamic Scheduling and Migration Framework for Cloud-Assisted Edge Clusters (No. arXiv:2501.07130). arXiv. <https://doi.org/10.48550/arXiv.2501.07130>
- [10]. Perera, H. C. S., De Silva, T. S. D., Wasala, W. M. D. C., Rajapakshe, R. M. P. R. L., Kodagoda, N., Samaratunge, U. S. S., & Jayanandana, H. H. N. C. (2021). Database Scaling on Kubernetes. 2021 3rd International Conference on Advancements in Computing (ICAC), 258 – 263. <https://doi.org/10.1109/ICAC54203.2021.9671185>
- [11]. Rudrabhatla, C. K. (2020). Comparison of zero downtime based deployment techniques in public cloud infrastructure. 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 1082 – 1086. <https://doi.org/10.1109/I-SMAC49090.2020.9243605>
- [12]. Senjab, K., Abbas, S., Ahmed, N., & Khan, A. ur R. (2023). A survey of Kubernetes scheduling algorithms. Journal of Cloud Computing, 12(1), 87. <https://doi.org/10.1186/s13677-023-00471-1>
- [13]. Sun, X., Luo, W., Gu, J. T., Ganesan, A., Alagappan, R., Gasch, M., Suresh, L., & Xu, T. (2022). Automatic Reliability Testing For Cluster Management Controllers. 143 – 159. <https://www.usenix.org/conference/osdi22/presentation/sun>
- [14]. Tran, M.-N., & Kim, Y. (2024). Optimized resource usage with hybrid auto-scaling system for knative serverless edge computing. Future Generation Computer Systems, 152, 304 – 316. <https://doi.org/10.1016/j.future.2023.11.010>
- [15]. Xu, Q., Gao, Y., & Wei, J. (2024). An Empirical Study on Kubernetes Operator Bugs. Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing

Tahira Amin



2018 September : B.S in
Computer Science,
COMSATS University
Islamabad, Pakistan
2024 March : M.S in
Computer Science,
COMSATS University

Islamabad, Pakistan

2025 September - Present: Ph.D. in Electronic
Engineering, Jeju National University, Republic
of Korea.

<Field of Interest> Natural Language Processing
(NLP), Large Language Models (LLMs),
Kubernetes and Container Orchestration,
Kubernetes Operators

Muhammad Asif



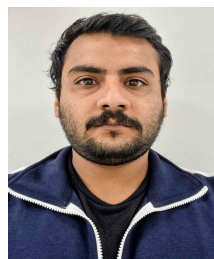
2015 September : B.S in
Computer Software
Engineering, Abasyn
University Peshawar, Pakistan
2020 September: M.S in
Computer Software
Engineering, University of

Engineering and Technology Peshawar, Pakistan

2014 January - Present: Ph.D. in Computer
Engineering, Jeju National University, Republic
of Korea.

<Field of Interest> Agentic AI-driven
autonomous and intent-based systems,
LLMs, NLP, STGNNs, zkML, cloud-native
computing with Kubernetes, SDN, secure and
privacy-preserving network automation, and
Space-Air-Ground Integrated Networks.

Umar Mahmood



2022 July: B.E, in Computer
Software Engineering.
National University of
Science and Technology,
Pakistan

2024 September - Present:
Ph.D. in Electronics

Engineering, Jeju National University, Republic
of Korea

<Field of Interest> Field of interest: computer
network, cloud networking, IBN, network
automation

Wang-cheol Song



1986 February: B.S in Food
Engineering, Yonsei
University, Republic of
Korea

1989 February: B.S in
Electronic Engineering,
Yonsei University, Republic

of Korea.

1991 February: M.S in Electronic Engineering,
Yonsei University, Republic of Korea.

1995 August: Ph.D. in Electronic Engineering,
Yonsei University, Republic of Korea.

<Field of Interest> VANETs and MANETs,
SDN/NFV, Intent-based networking, and Network
management.