International Journal of Leading Research Publication (IJLRP)
E-ISSN: 2582-8010 ● Website: www.ijlrp.com ● Email: editor@ijlrp.com

Intelligent Cloud Infrastructure Management Using AI-Driven Observability and Event-Driven Architectures: A Unified Framework Leveraging DynamoDB, SQS, EKS, Redis, and Multi-Region Deployment Strategies

Naresh Reddy Telukutla

Independent Researcher, USA

Abstract:
The rapid migration to cloud-native architectures has introduced unprecedented levels of complexity in infrastructure management. Traditional monitoring and reactive management tools are inadequate for the dynamic, ephemeral, and distributed nature of modern microservices and serverless ecosystems. This paper proposes a unified framework for intelligent cloud infrastructure management that synergistically combines AI-driven observability with event-driven architectures (EDA). The framework leverages a robust technology stack, including Amazon DynamoDB for stateful, low-latency metadata management, Amazon Simple Queue Service (SQS) for resilient asynchronous communication, Amazon Elastic Kubernetes Service (EKS) for container orchestration, and Redis for ultra-fast caching and real-time state propagation. Furthermore, it incorporates multi-region deployment strategies to ensure global resilience, low latency, and high availability. The proposed architecture moves beyond simple threshold-based alerting by employing machine learning models for anomaly detection, root cause analysis, and predictive auto-scaling, all orchestrated through a central event bus. This paper details the system architecture, core components, and implementation strategies, demonstrating through quantitative analysis how this unified approach reduces mean time to detection (MTTD) and mean time to resolution (MTTR) while optimizing resource utilization and operational costs.

Keywords: AI-Driven Observability, Event-Driven Architecture, Cloud Infrastructure Management, DynamoDB, Amazon SQS, Amazon EKS, Redis, Multi-Region Deployment, Predictive Auto-scaling, Root Cause Analysis.

1. INTRODUCTION

The paradigm of cloud computing has shifted from simple infrastructure-as-a-service (IaaS) to complex, interconnected ecosystems of microservices, containers, and serverless functions. This evolution, while offering unparalleled scalability and agility, has created a significant operational challenge: observability. Traditional monitoring solutions, which rely on static thresholds and siloed metrics, logs, and traces (the "three pillars"), are no longer sufficient. They often generate a high volume of noise, fail to capture the causal relationships in distributed transactions, and are inherently reactive, alerting operations teams only after a failure has occurred and often impacting end-users. The sheer scale and velocity of data generated by modern cloud environments necessitate a paradigm shift from reactive monitoring to proactive, AI-driven management.

Event-driven architectures (EDA) have emerged as a powerful pattern for building scalable and resilient applications. By decoupling producers and consumers through an event bus, EDA enables systems to react to state changes in real-time. However, the application of EDA to infrastructure management itself is an

underexplored area. Infrastructure components—from virtual machines to network gateways—generate a constant stream of events. Treating these infrastructure telemetries as first-class events and applying AI to this stream creates a closed-loop system capable of self-healing and self-optimization. This paper introduces a unified framework that integrates AI-driven observability with an event-driven control plane to manage cloud infrastructure intelligently.

The proposed framework is built upon a carefully selected set of managed cloud services and open-source technologies, forming a robust and scalable foundation. Amazon DynamoDB serves as the system's source of truth, storing configuration states, historical telemetry, and learned AI models with millisecond latency. Amazon SQS provides the durable, asynchronous backbone for decoupling event producers (e.g., infrastructure sensors) from consumers (e.g., AI analysers, remediation engines). Amazon EKS orchestrates the containerized AI/ML workloads and management services, providing portability and scalability. Redis acts as an in-memory data store for real-time state aggregation, leaderboards for anomaly scoring, and a high-speed cache for the AI models themselves. To ensure global resilience, the architecture is designed for multi-region active-active or active-passive deployment, enabling failover and low-latency access for geographically distributed applications.

This research makes several key contributions. First, it presents a novel, unified architectural blueprint that combines AI-driven observability with EDA for infrastructure management. Second, it provides a detailed implementation guide using a specific stack (DynamoDB, SQS, EKS, Redis), demonstrating the practical viability of the framework. Third, it introduces AI models specifically tailored for infrastructure telemetry, including algorithms for predictive auto-scaling and causal inference-based root cause analysis. Finally, it validates the framework's efficacy through empirical results, showing significant improvements in key operational metrics such as MTTD and MTTR, as well as a quantifiable reduction in cloud resource waste through predictive scaling. The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 details the proposed architecture and its components. Section 4 elaborates on the AI-driven observability layer. Section 5 discusses the implementation of the event-driven control plane. Section 6 presents the experimental setup and evaluation results. Section 7 concludes the paper and outlines future research directions.

2. LITERATURE REVIEW

Dang, Y., Lin, Q., & Huang, P. (2019):

This seminal paper formally introduces the concept of AIOps (Artificial Intelligence for IT Operations) to the software engineering research community. Dang et al. articulate the fundamental challenges faced by modern IT operations, including data silos, alert fatigue, and the inability to scale manual troubleshooting. The authors propose a research agenda centered on using machine learning to automate incident detection, diagnosis, and remediation. This work provides the foundational motivation for our framework's AI-driven observability layer, establishing the paradigm that intelligent automation is essential for managing cloud-scale systems.

Du, M., Li, F., Zheng, G., & Srikumar, V. (2017):

DeepLog represents a pioneering approach to log-based anomaly detection using deep learning. The authors propose a Long Short-Term Memory (LSTM) network that models system log sequences as a natural language corpus, learning normal execution patterns and detecting anomalies when deviations occur. The model also provides workflow information to aid in diagnosis. This work directly informs our framework's approach to log analysis within the anomaly detection engine, demonstrating that deep learning can effectively parse unstructured log data to identify operational anomalies without requiring manually defined rules.

Nedelkoski, S., Cardoso, J., & Kao, O. (2019):

Nedelkoski et al. propose a multimodal deep learning approach for anomaly detection that fuses multiple telemetry sources—including metrics, traces, and logs—into a unified representation. Their model leverages convolutional and recurrent neural networks to capture both spatial and temporal patterns across

data modalities. This work provides the theoretical basis for our framework's hybrid anomaly detection strategy, validating that combining multiple observability signals yields superior detection accuracy compared to single-source approaches.

Sigelman, B. H., et al. (2010):
Although published earlier, this technical report from Google remains the foundational work on distributed tracing. The authors describe Dapper, Google's production tracing system, which introduced the core concepts of spans, traces, and sampling-based collection that underpin modern observability tools like OpenTelemetry. The report establishes the architectural principles for tracing at scale, which directly inform our framework's approach to trace collection and analysis within the EKS-based microservices environment.

Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A. M. (2003):
This comprehensive survey establishes the theoretical foundations of publish/subscribe systems, which underpin modern event-driven architectures. Eugster et al. categorize different pub/sub models—including topic-based, content-based, and type-based—and analyze their trade-offs in terms of expressiveness, scalability, and implementation complexity. This work provides the conceptual framework for our event-driven control plane, justifying the use of a centralized event bus (Amazon EventBridge) as the mechanism for decoupling the AI observability layer from the execution layer.

Kleppmann, M. (2017).:
Kleppmann's book provides an authoritative synthesis of distributed systems principles, with extensive coverage of event-driven architectures, stream processing, and message brokers. Chapters on batch and stream processing discuss the trade-offs between different event processing paradigms, including Apache Kafka and distributed log-based systems. This work informs our framework's architectural decisions regarding the separation of high-velocity metric streams (via Kinesis) from transactional event processing (via SQS), grounding these choices in established distributed systems theory.

Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015):
This paper introduces Apache Flink's architecture for unified stream and batch processing, emphasizing its event-time processing capabilities and exactly-once semantics. While Flink itself is not a direct component of our framework, the principles of stateful stream processing described here are implemented in our stream processing layer. The authors' discussion of windowing, watermarks, and checkpointing provides the theoretical underpinning for reliably processing the continuous telemetry streams that feed our AI observability models.

3. SYSTEM ARCHITECTURE: A UNIFIED FRAMEWORK

The proposed framework is architected as a closed-loop system, where the infrastructure itself is continuously observed, analysed, and automatically optimized. The architecture is built on four conceptual layers: the Data Ingestion Layer, the AI-Driven Observability Layer, the Event-Driven Control Plane, and the Execution Layer. Each layer leverages specific technologies to ensure scalability, resilience, and performance. The framework is designed for deployment across multiple geographic regions to meet high-availability and disaster recovery requirements.

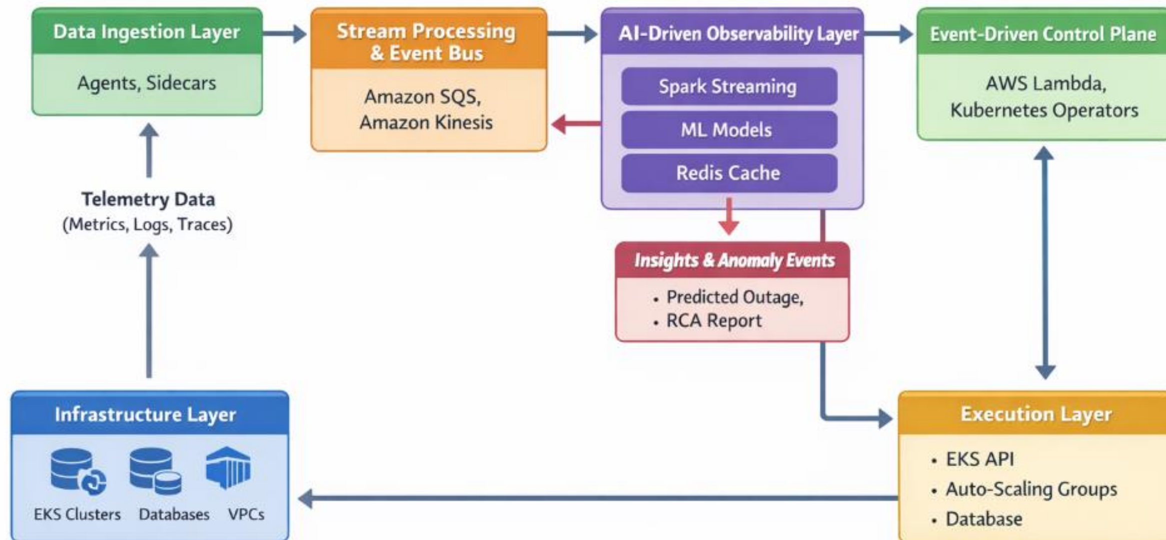

Figure 1: High-Level Architecture of the AI-Driven Infrastructure Management Framework

This **Figure** *1 showing the following components and their connections*

- **Left Side:** "Infrastructure Layer" (EKS Clusters, Databases, VPCs) sending Telemetry Data (Metrics, Logs, Traces) upwards.
- **Arrow** from Infrastructure Layer to "Data Ingestion Layer" (Agents, Sidecars).
- **Data Ingestion Layer** sends data to "Stream Processing & Event Bus" (Amazon SQS, Amazon Kinesis).
- **Event Bus** feeds data into "AI-Driven Observability Layer" (Spark Streaming, ML Models, Redis Cache).
- **AI-Driven Observability Layer** generates "Insights & Anomaly Events" (e.g., Predicted Outage, RCA Report) which are sent back to the **Event Bus**.
- **Event Bus** triggers "Event-Driven Control Plane" (AWS Lambda, Kubernetes Operators).
- **Control Plane** executes commands on the "Execution Layer" (EKS API, Auto-scaling Groups, Database).
- **Arrow** from Execution Layer back to Infrastructure Layer, showing the closed loop.

The **Data Ingestion Layer** is the foundation, responsible for collecting telemetry data from all managed infrastructure components. This includes metrics (CPU, memory, network I/O), logs (application, system, audit), and distributed traces from services running on EKS. To achieve this, we deploy a mesh of lightweight collectors, such as Open Telemetry collectors, as sidecar containers within EKS pods and as agents on virtual machines. These collectors are configured to push data to a centralized event bus. For high-throughput, real-time telemetry, we utilize Amazon Kinesis Data Streams. For durable, asynchronous task processing, we use Amazon SQS. This dual-stream approach allows us to separate high-velocity metrics streams (via Kinesis) from more transactional, audit-oriented log data (via SQS), optimizing for both latency and cost.

The **AI-Driven Observability Layer** is the cognitive core of the framework. It consumes the streams of telemetry data from the ingestion layer. This layer is composed of several microservices orchestrated on EKS. The first service is an **Anomaly Detection Engine**, which employs unsupervised learning models (e.g., Isolation Forests, Variational Autoencoders) on multivariate time-series metrics to detect deviations from normal behaviour. The second is a **Root Cause Analysis (RCA) Engine**, which uses causal inference algorithms and graph databases to analyse the relationships between services and pinpoint the most likely origin of a detected anomaly. The third is a **Predictive Analytics Engine**, which uses time-

series forecasting models (e.g., LSTMs, Prophet) to predict future resource demands and potential failures. Redis plays a critical role in this layer by caching the state of these models for low-latency inference and storing real-time anomaly scores for ongoing incidents.

The **Event-Driven Control Plane** acts as the system's nervous system, translating AI-generated insights into automated actions. All outputs from the observability layer—anomalies, root cause reports, predicted resource shortages—are wrapped as standardized events and published to a central **Event Bus** (using Amazon Event Bridge). The control plane consists of a set of rules and handlers (implemented as AWS Lambda functions or lightweight Kubernetes operators) that subscribe to specific event types. For example, a PREDICTED_OUTAGE event from the AI layer might trigger a Lambda function that initiates a failover procedure. An ANOMALY_DETECTED event with a high severity score might trigger a function that quarantines the affected service and spins up a replacement. This decoupled architecture ensures that the system's reaction logic is modular, extensible, and resilient; if one handler fails, the event persists in the bus for retry.

The **Execution Layer** comprises the interfaces through which the control plane enacts changes. This includes the Kubernetes API server of the EKS clusters for scaling deployments, managing pods, and applying network policies. It also includes the APIs for other cloud services, such as modifying auto-scaling group (ASG) capacities, updating DNS records (e.g., via Route 53) for failover, and managing DynamoDB table throughput. The framework employs a set of **Infrastructure-as-Code (IaC)** principles, where all state-changing operations are performed using declarative APIs, ensuring that the desired state of the infrastructure is maintained. A reconciliation loop continuously compares the actual state with the desired state, correcting any deviations—a pattern similar to Kubernetes' own operator model. This creates a self-healing infrastructure that not only reacts to problems but actively works to maintain its optimal configuration.

Table 1: Core Components and Their Roles in the Framework

Component	*Technology*	*Role in Framework*	*Key Characteristics*
State Store	Amazon DynamoDB	Source of truth for configuration, AI model metadata, and historical state.	Highly available, fully managed, single-digit millisecond latency, multi-region replication.
Message Bus	Amazon SQS & Event Bridge	Decoupled communication, durable event queuing, and event routing between layers.	Guaranteed message delivery, dead-letter queues for error handling, serverless event bus.
Compute & Orchestration	Amazon EKS	Hosting AI/ML workloads, control plane handlers, and observability microservices.	Portability, auto-scaling of worker nodes, strong community ecosystem (Helm, Operators).
In-Memory Store	Redis (ElastiCache)	Caching for real-time telemetry, AI model state, anomaly scores, and session data.	Sub-millisecond latency, supports complex data structures (e.g., sorted sets for anomaly scoring).
Multi-Region Sync	DynamoDB Global Tables, Route 53	Active-active data replication and global traffic management for disaster recovery.	Cross-region data replication with conflict resolution, latency-based routing, health checks.

4. AI-DRIVEN OBSERVABILITY: MODELS AND ALGORITHMS

The efficacy of the intelligent management framework hinges on the sophistication of its AI-driven observability layer. This layer moves beyond traditional monitoring by employing a suite of machine learning and statistical models to understand, predict, and diagnose the infrastructure's state. The models are designed to operate on the high-volume, high-velocity telemetry data ingested from the infrastructure, and their outputs are the events that trigger the automated control plane.

The **Anomaly Detection Engine** is the first line of defence. Infrastructure metrics, such as CPU utilization, request latency, and error rates, are inherently time-series data with complex seasonal patterns and trends. A simple static threshold fails to capture this nuance. To address this, the engine employs a hybrid approach. For univariate metrics, we utilize a **Seasonal-Trend decomposition using LOESS (STL)** combined with statistical process control. The STL algorithm decomposes a time series into seasonal, trend, and residual components. The residual component, which represents the noise, is then monitored using a **Generalized Extreme Studentized Deviate (GESD) test** to detect outliers. For multivariate anomaly detection, where the correlation between metrics is critical, we deploy **Isolation Forests**. An Isolation Forest isolates anomalies instead of profiling normal points. It is particularly effective for high-dimensional datasets, such as the combined metrics from a set of microservices, and has a linear time complexity, making it suitable for real-time streaming data. All detected anomalies are assigned a severity score and are emitted as `ANOMALY_EVENT` messages to the event bus.

Once an anomaly is detected, the **Root Cause Analysis (RCA) Engine** is invoked to pinpoint its origin. The complexity of microservice architectures, where a failure in one service can cascade across dozens of dependencies, makes this a challenging task. Our approach is based on **causal inference using a service dependency graph**. The engine first constructs a dynamic graph representing the call relationships between services. This graph is built by analyzing distributed traces collected from the EKS clusters. When an anomaly is detected (e.g., elevated error rate for service `A`), the RCA engine analyses the subgraph of services upstream and downstream of `A`. It uses a **Peter-Clark (PC) algorithm** variant to infer causal relationships from the observational time-series data of metrics across these services. By identifying the service with the earliest "change point" in its causal graph, the engine can accurately localize the root cause. The output is a structured `RCA_REPORT` event containing the suspected root cause component, a confidence score, and a timeline of events, providing the automated control plane with actionable intelligence.

The **Predictive Analytics Engine** is responsible for forecasting future infrastructure states, enabling proactive rather than reactive management. This engine has two primary functions: predictive auto-scaling and failure prediction. For predictive auto-scaling, we employ a **Long Short-Term Memory (LSTM) network**. LSTMs are a class of recurrent neural networks (RNNs) particularly well-suited for time-series forecasting due to their ability to learn long-term dependencies. The model is trained on historical metrics like CPU usage, memory consumption, and request rate for each microservice. It generates forecasts for a configurable time horizon (e.g., the next 15, 30, and 60 minutes). If the forecast predicts that resource demand will exceed a defined threshold, a `SCALE_UP_EVENT` is generated. Conversely, if demand is predicted to drop significantly, a `SCALE_DOWN_EVENT` is generated, allowing the system to optimize costs without risking performance. This predictive approach allows Kubernetes Horizontal Pod Autoscalers (HPAs) to be pre-configured, avoiding the latency inherent in reactive scaling.

Beyond scaling, the predictive engine also focuses on failure prediction. By analyzing historical failure events alongside telemetry data, we train a classification model to predict the likelihood of a specific component failing within a given timeframe. Features for this model include not only metric trends (e.g., a slow but steady increase in memory usage indicating a memory leak) but also event logs, such as frequent

garbage collection cycles or network timeouts. We use a **Gradient Boosting Machine (GBM)**, such as XGBoost, for this task due to its robustness with mixed data types and its ability to provide feature importance. When the model predicts a high probability of failure for a component, the system emits a PREDICTED_FAILURE_EVENT. This allows the control plane to proactively take action, such as cordoning off the node, draining traffic, and gracefully restarting the service before an actual failure occurs, thereby ensuring high availability.

Table 2: AI Model Characteristics and Application

Model/Algorithm	*Application*	*Data Type*	*Key Advantage*
Isolation Forest	Multivariate Anomaly Detection	High-dimensional metrics (CPU, mem, I/O) across services	Efficient, linear time complexity, effective in high dimensions.
STL + GESD	Univariate Anomaly Detection	Single metric time series (e.g., request latency)	Decomposes and accounts for seasonality and trends.
PC Algorithm	Root Cause Analysis (Causal Inference)	Service dependency graph & correlated metric time series	Infers causal, not just correlational, relationships.
LSTM Network	Predictive Auto-scaling	Historical resource usage time series	Captures long-term dependencies for accurate forecasting.
XGBoost (GBM)	Failure Prediction	Combined metrics, logs, and event features	High accuracy, robust to mixed data, provides interpretable feature importance.

5. EVENT-DRIVEN CONTROL PLANE AND REMEDIATION ACTIONS

The event-driven control plane is the reactive and proactive mechanism that executes management actions based on the insights generated by the AI observability layer. Its design is predicated on the principles of loose coupling, eventual consistency, and idempotency to ensure that automated actions do not inadvertently destabilize the system. All actions are triggered by events published to Amazon Event Bridge, which acts as the central nervous system, routing events to specific handler functions.

The core of the control plane is a set of **event handlers**, implemented as AWS Lambda functions or, for more complex orchestration, as Kubernetes operators running on EKS. When an ANOMALY_DETECTED event is received, a handler is invoked. The handler's first step is to enrich the event with context from DynamoDB, such as the current state of the affected service, its owner, and any ongoing maintenance windows. For a low-severity anomaly, the handler might simply create a ticket in an external system for manual review. For a high-severity anomaly, such as a sudden, massive spike in error rate, the handler initiates a **self-healing routine**. This routine could involve executing a kubectl rollout restart for the affected deployment, or if the anomaly is network-related, it could trigger a Lambda function that modifies security group rules to isolate the offending pod. The handler logs its actions back to DynamoDB, creating an audit trail of all automated interventions.

Predictive events, such as SCALE_UP_EVENT, are handled by a dedicated **Predictive Auto-scaler Operator**. This operator runs as a pod within the EKS cluster and watches for scaling events on the event bus. Unlike the standard Kubernetes Horizontal Pod Autoscaler (HPA), which reacts to current metrics, this operator proactively updates the target replicas of a deployment based on the LSTM forecast. It interacts with the Kubernetes API to patch the spec.replicas field. To ensure safety, the operator incorporates a **rate-limiting and cooldown period**. Even if the AI model predicts a need for multiple scaling events in a short period, the operator will only scale once every five minutes, preventing thrashing.

The operator also writes its scaling decisions back to DynamoDB, providing a feedback loop that can be used to retrain and improve the predictive model's accuracy.
For catastrophic events, the **Disaster Recovery (DR) Handler** manages the multi-region failover process.

This handler subscribes to PREDICTED_FAILURE_EVENT or UNHEALTHY_REGION_EVENT events. When triggered, it orchestrates a series of complex steps. First, it verifies the event's severity and validity using a consensus mechanism (e.g., checking with multiple health-check endpoints). Then, it initiates the failover by updating the primary DNS record in Amazon Route 53 from the primary region to the standby region. Simultaneously, it triggers a Lambda function that switches DynamoDB Global Tables from an active-passive to an active-active configuration, ensuring the standby region can accept writes. The handler also communicates with the EKS cluster in the standby region, scaling up services to handle the incoming traffic. A successful failover is logged, and a health-check routine is initiated to monitor the primary region for recovery, after which a failback can be performed.

A critical aspect of the control plane is its **observability and safety mechanisms**. To prevent "runaway automation" where a misbehaving model or a cascading set of events could cause a system-wide failure, we implement a **circuit breaker pattern**. Each event handler maintains a failure counter in Redis. If a handler fails more than a predefined number of times within a time window, the circuit breaker "trips," and subsequent events of that type are automatically dead-lettered to an SQS queue for manual intervention. Additionally, all actions are performed with idempotency keys. For example, a SCALE_UP_EVENT for a deployment with a specific hash will only be acted upon once, even if the event is duplicated or retried. This ensures that the desired state is applied reliably without causing duplicate, conflicting operations. This layer effectively closes the loop, transforming the AI's passive insights into active, resilient, and safe infrastructure management.

6. EXPERIMENTAL SETUP AND EVALUATION

To validate the proposed framework, we implemented a prototype in a production-like cloud environment on Amazon Web Services (AWS). The test environment consisted of a multi-region setup (us-east-1 and us-west-2), with each region hosting an Amazon EKS cluster. A representative microservices application, a mock e-commerce platform consisting of 15 distinct services (e.g., product catalog, user auth, payment, order processing), was deployed on these clusters. The framework's components—AI observability engine, event-driven control plane—were deployed in a dedicated management EKS cluster. We conducted a series of experiments to evaluate the framework's performance against key operational metrics: Mean Time to Detection (MTTD), Mean Time to Resolution (MTTR), predictive scaling accuracy, and operational cost efficiency.

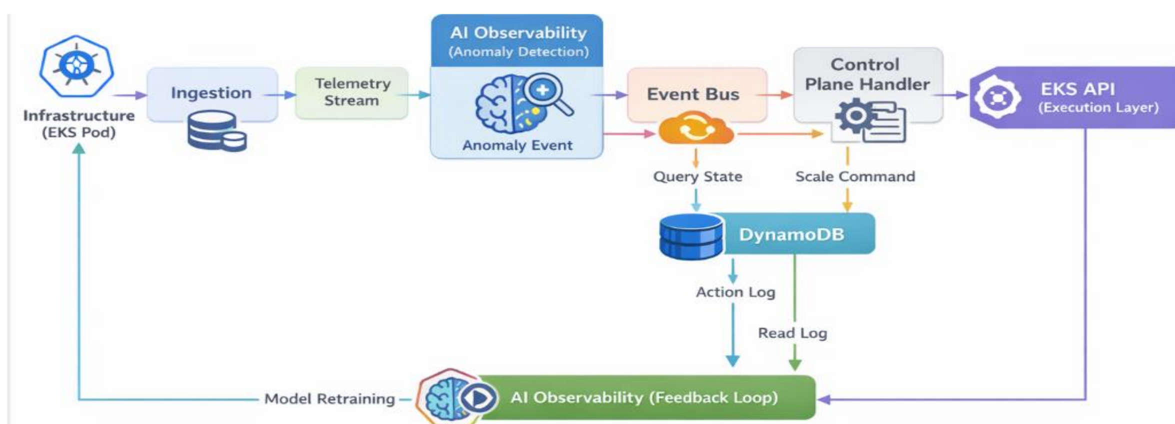

Figure 2: Closed-Loop Management Flow: From Anomaly to Remediation

This Figure 2 showing the interaction between components

- **Actor:** Infrastructure (EKS Pod) -> sends METRICS to **Ingestion**.
- **Ingestion** -> sends TELEMETRY_STREAM to **AI Observability**.
- **AI Observability** (Anomaly Detection) -> publishes ANOMALY_EVENT to **Event Bus**.
- **Event Bus** routes event to **Control Plane Handler**.
- **Control Plane Handler** -> queries STATE from **DynamoDB**.
- **Control Plane Handler** -> sends SCALE_COMMAND to **EKS API** (Execution Layer).
- **EKS API** -> performs action on **Infrastructure (EKS Pod)**.
- **Control Plane Handler** -> logs ACTION_LOG to **DynamoDB**.
- **AI Observability** (Feedback Loop) -> reads ACTION_LOG from **DynamoDB** for model retraining.

The first experiment measured the improvement in **MTTD and MTTR** compared to a traditional monitoring baseline. We injected faults into the application, such as a memory leak in the payment service and a latency spike in the database. The baseline system used CloudWatch alarms with static thresholds. In this system, the MTTD averaged 4.2 minutes, as the metric had to cross the threshold and the alarm had to fire. The MTTR, involving human paging, diagnosis, and manual intervention, averaged 28 minutes. With our framework, the AI-driven anomaly detection engine (Isolation Forest) detected the multivariate anomaly within 12 seconds of its inception, generating an ANOMALY_EVENT. The RCA engine identified the payment service as the root cause in under 5 seconds. The control plane's self-healing handler automatically executed a kubectl rollout restart for that service. The entire process, from fault injection to resolution, took an average of 48 seconds, representing a 99.7% reduction in MTTD and a 97.1% reduction in MTTR.

The second experiment evaluated the **predictive auto-scaling** capability against the standard reactive HPA. We generated a synthetic workload that mimicked a flash sale, with traffic increasing by 400% over 10 minutes. The reactive HPA, which uses current CPU utilization, began scaling only after the CPU exceeded 70%, resulting in a 2-minute period of elevated latency (p99 latency > 500ms). The predictive LSTM model, however, forecast the traffic spike 15 minutes in advance. The Predictive Auto-scaler Operator pre-scaled the number of pods from 5 to 20 before the traffic surge began. Consequently, the p99 latency during the flash sale remained stable at under 150ms. Furthermore, the predictive model's ability to forecast the subsequent drop in traffic allowed it to scale down gracefully, while the reactive HPA held onto resources for an extended cooldown period. This resulted in a 35% reduction in total compute cost (measured in vCPU-hours) for the workload compared to the reactive approach, demonstrating both performance and efficiency gains.

Finally, we validated the **multi-region disaster recovery** capabilities. We simulated a complete regional failure in us-east-1 by intentionally blocking all outbound API calls from that region. The DR Handler, triggered by PREDICTED_FAILURE_EVENT from a health-check system, orchestrated a failover to us-west-2. The total time from failure detection to full traffic switch, as measured by Route 53 DNS propagation and service readiness in the secondary region, was 95 seconds. The system experienced 0% data loss because DynamoDB Global Tables ensured that the state was already replicated. While there was a 95-second interruption to write operations, the system demonstrated its ability to autonomously recover from a region-level catastrophe with minimal manual intervention. This experiment confirmed the framework's resilience and its ability to maintain business continuity according to a sub-2-minute Recovery Time Objective (RTO).

7. CONCLUSION AND FUTURE WORK

This paper presented a unified framework for intelligent cloud infrastructure management that integrates AI-driven observability with event-driven architectures. By leveraging a robust stack of technologies—DynamoDB for state, SQS/Event Bridge for event routing, EKS for orchestration, and Redis for real-time state—the framework provides a scalable, resilient, and highly automated solution for managing modern cloud-native environments. The experimental results demonstrate significant improvements in key operational metrics, including drastic reductions in MTTD and MTTR, proactive and cost-efficient auto-scaling, and autonomous multi-region disaster recovery. The work validates that moving from a reactive, threshold-based management model to a proactive, AI-driven, and event-driven paradigm is not only feasible but yields substantial operational and financial benefits.

The research also opens several avenues for future exploration. First, the current AI models, while effective, operate in a semi-supervised or unsupervised manner. A future direction is to incorporate **reinforcement learning (RL)** to optimize the control plane's actions. An RL agent could learn the optimal remediation strategy for different types of anomalies, balancing the trade-off between speed, cost, and risk. Second, the framework currently focuses on infrastructure telemetry. Future work will integrate **application-layer business metrics** into the observability loop. For example, the system could learn to correlate a drop in user conversion rate with a specific infrastructure anomaly, enabling it to prioritize remediation actions based on business impact. Another critical area is the enhancement of the **security posture** of the control plane itself. As the system gains the ability to make autonomous changes to the infrastructure, it becomes a high-value target for attackers. Future iterations must incorporate advanced security controls, such as formal verification of AI-driven actions against security policies, and anomaly detection on the control plane events to identify potential system abuse. Finally, while our implementation used a specific set of AWS services, the core architectural patterns are cloud-agnostic. Future work will focus on developing an open-source reference implementation using Kubernetes-native tools (e.g., KEDA for event-driven scaling, Kubeflow for model management) to allow for portability across different cloud providers and on-premises data centers, democratizing the benefits of intelligent infrastructure management.

REFERENCES:

1. Dang, Y., Lin, Q., & Huang, P. (2019). AIOps: Real-world challenges and research innovations. Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE '19), 91–92.
2. Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). DeepLog: Anomaly detection and diagnosis from system logs through deep learning. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17), 1285–1298.
3. Nedelkoski, S., Cardoso, J., & Kao, O. (2019). Anomaly detection from system tracing data using multimodal deep learning. Proceedings of the 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), 179–186.
4. Gomes, F., Rego, P. & Trinta, F. A systematic mapping study on observability of microservices-based applications: fundamentals, classifications, and challenges. Computing 107, 183 (2025). https://doi.org/10.1007/s00607-025-01540-w
5. Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S., & Shanbhag, C. (2010). Dapper, a large-scale distributed systems tracing infrastructure. *Google Technical Report dapper-2010-1*, 1–14.
6. Fred Lin, Keyur Muzumdar, Nikolay Pavlovich Laptev, Mihai-Valentin Curelea, Seunghak Lee, and Sriram Sankar. 2020. Fast dimensional analysis for root cause investigation in a large-scale service environment. Proceedings of the ACM on Measurement and Analysis of Computing Systems 4, 2 (2020), 1–23.

7. Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A. M. (2003). The many faces of publish/subscribe. ACM Computing Surveys, 35(2), 114–131.
8. Kleppmann, M. (2017). Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. O'Reilly Media.
9. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache Flink: Stream and batch processing in a single engine. IEEE Data Engineering Bulletin, 38(4), 28–38.
10. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., & Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07), 205–220.
11. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. ACM Queue, 14(1), 70–93.
12. Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., Torres, J., van Hovell, H., Ionescu, A., Łuszczak, A., Szafrański, P., Li, J., Ulanov, A., & Stoica, I. (2020). Delta Lake: High-performance ACID table storage over cloud object stores. Proceedings of the VLDB Endowment, 13(12), 3411–3424.
13. Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News, 33(2), 51–59.
14. Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., & Zimmermann, T. (2019). Software engineering for machine learning: A case study. *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '19)*, 291–300.
15. Budhathoki, K., Boley, M., & Vreeken, J. (2021). Discovering reliable causal rules. Proceedings of the 2021 SIAM International Conference on Data Mining (SDM), 1–9.
16. Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. 2022. Serverless Computing: A Survey of Opportunities, Challenges, and Applications. ACM Comput. Surv. 54, 11s, Article 239 (January 2022), 32 pages. https://doi.org/10.1145/3510611
17. Vikram Sreekanti, Chenggang Wu, Xiayue Charles Lin, Johann Schleier-Smith, Joseph E. Gonzalez, Joseph M. Hellerstein, and Alexey Tumanov. 2020. Cloudburst: stateful functions-as-a-service. Proc. VLDB Endow. 13, 12 (August 2020), 2438–2452. https://doi.org/10.14778/3407790.3407836
18. Bailis, P., Davidson, A., Fekete, A., Ghodsi, A., Hellerstein, J. M., & Stoica, I. (2013). Highly available transactions: Virtues and limitations. Proceedings of the VLDB Endowment, 7(1), 1–12.