Journal of Information Systems Engineering and Management
2026, 11(1s) e-ISSN:
2468-4376
https://www.jisem-journal.com/

Research Article

Predictive Autoscaling in Kubernetes Microservices with KEDA and Time Series Forecasting

Tina Lekshmi Kanth
Illinois Institute of Technology, Illinois, USA

ARTICLE INFO

Received: 19 Dec 2025

Revised: 24 Dec 2025

ABSTRACT

Cloud-native microservices architectures handling large volumes of transactions require smart resource management mechanisms that go beyond traditional reactive autoscaling boundaries. Threshold-based scaling is traditional, bringing with it the natural latency between demand changes and capacity changes, leading to compromised performance during traffic spikes and wastage of resources during dips in demand. Event-driven autoscaling systems go beyond infrastructure-level metrics by involving external data feeds, message queue sizes, and application-level metrics for making scaling decisions. Yet, reactive mechanisms are inherently bounded by response latencies that degrade service quality and operational effectiveness. Predictive autoscaling bridges such gaps with time series forecasting models that interpret historical workload patterns to predict future resource needs. The combination of deep neural network architectures with event-driven autoscaling modules allows for proactive capacity provisioning in response to projected workload trends as opposed to observed metric values. Multivariate prediction models with correlated resource metrics are superior in prediction accuracy over univariate methods, able to capture intricate interdependencies between CPU usage, memory usage, network bandwidth, and storage activities. Design considerations include model choice based on workload behaviour, hyperparameter tuning to balance accuracy with computational cost, and reliable integration frameworks with extensive error handling and fallback strategies. Predictive approaches show considerable benefits such as lower response time degradation, better cost-effectiveness with enhanced resource utilisation, and reduced scaling operation frequency. The challenges are to ensure accuracy of the forecast in the face of changing traffic patterns, address computational overhead due to periodic model retraining, and provide tolerance to prediction uncertainties affecting the scaling aggressiveness.

Keywords: Predictive Autoscaling, Event-Driven Scaling, Time Series Forecasting, Deep Neural Networks, Cloud Resource Management, Microservices Elasticity

I. INTRODUCTION

Cloud-native applications developed in modern times on microservice architectures are confronted with resource management and scalability challenges like never before. These distributed systems that handle enormous amounts of transactions every day need high-level, sophisticated mechanisms to ensure maximum performance while managing operational expenses. Applications built with gRPC APIs execute business logic and persist transactions in document stores such as Cosmos DB, while simultaneously publishing events to Azure Event Hub for downstream processing. These systems experience traffic variability influenced by temporal factors such as time of day, day of week, and seasonal patterns, necessitating intelligent autoscaling strategies that can anticipate demand rather than merely react to it.

The development of container orchestration platforms has introduced Kubernetes Event-Driven Autoscaling (KEDA) as an advanced autoscaling mechanism beyond traditional CPU and memory-based approaches. While the native Horizontal Pod Autoscaler operates via periodic metric requests at specified intervals and scales replica numbers according to preset utilisation levels, KEDA extends this functionality by enabling workload scaling based on external event sources and metrics. KEDA supports multiple scaler types including message queue depths from Azure Event Hub and Kafka, database connection pools, custom application metrics from Prometheus, and HTTP endpoint responses, allowing organisations to scale based on business-relevant metrics rather than solely infrastructure utilisation. This capability is particularly valuable for microservices architectures where applicationlevel metrics such as event hub message backlog or database transaction rates provide more meaningful scaling signals than generic CPU or memory thresholds.

However, KEDA's reactive autoscaling mechanisms necessarily introduce latency between changes in demand and adjustments in resources, causing potential performance decline during traffic peaks or wasteful over-provisioning during expected increases in load. Experiments have shown that traffic-informed autoscaling policies based on realtime traffic monitoring and predictive analysis are able to provide much better resource use and response times than the conventional threshold-based policies [5]. The traditional reactive scaling mechanisms wait for metrics to cross thresholds before initiating scaling operations, resulting in a gap between when capacity is needed and when it becomes available. Research comparing Kubernetes-based edge computing infrastructure has demonstrated that traffic-aware horizontal pod autoscaling mechanisms significantly outperform traditional reactive approaches, with experimental results showing that predictive policies reduce service level agreement breaches and improve resource usage efficiency by anticipating workload changes rather than merely reacting to threshold violations [5].

Event-driven autoscaling goes beyond conventional methods by allowing for workload scaling based on outside metrics and event sources in addition to measurements at the infrastructure level. These higher-level frameworks accommodate various scaler types such as message queue depths, database connection pools, custom app metrics, and HTTP endpoint responses so that organisations can scale on business-relevant metrics instead of just infrastructure utilisation. The architecture allows for scaling to zero replicas during times of inactivity, a feature highly beneficial in development environments and batch processing workloads where persistent resource provisioning is too expensive. Distributed microservice research has proven that adaptive load balancing combined with knowledge-driven autoscaling algorithms can maximise resource utilisation by automatically adjusting computational resources according to real-time demand patterns and forecasting analytics [2]. Such adaptive algorithms include machine learning methods for dissecting past traffic patterns, anticipating future workload trends, and dynamically adjusting resource allocation before demand realisation. Research has indicated that microservice designs using adaptive autoscaling solutions result in better response times, lower costs of operations, and increased system dependability than static provisioning strategies [2]. Augmenting event-driven scaling with predictive analytics allows systems to anticipate traffic variations based on temporal patterns, seasonality, and historical data, addressing the inherent limitations of reactive practices that act only after resource bottlenecks emerge.

This work introduces an end-to-end framework to deploy predictive autoscaling in Kubernetes microservices environments by coupling time series forecasting models with KEDA's event-driven autoscaling capabilities. The solution allows for proactive resource allocation based on analysis of traffic patterns from gRPC API request rates, Cosmos DB transaction volumes, and Azure Event Hub message throughput, offering the adaptability of KEDA's external metric-based scaling combined with statistical forecasting techniques' foresight. By analysing historical traffic patterns and generating forward-looking predictions, the framework addresses the inherent latency of reactive scaling while maintaining the flexibility and extensibility of KEDA's autoscaling platform. The integration takes advantage of KEDA's HTTP scaler functionality to ingest predicted metrics as external data sources that inform scaling decisions in anticipation of realised demand, ensuring service performance is maintained during traffic spikes and resources are optimally utilised during variable-demand periods. KEDA's architecture also allows for scaling to zero replicas during times of inactivity, a feature highly beneficial in development environments and batch processing workloads where persistent resource provisioning incurs unnecessary costs.

II. BACKGROUND AND BASIC IDEAS

A. Event-Driven Autoscaling with KEDA

Kubernetes Event-Driven Autoscaling (KEDA) is an open-source component that extends Kubernetes' native autoscaling capabilities by enabling workloads to scale based on external event sources and metrics beyond traditional CPU and memory utilisation. Unlike the Horizontal Pod Autoscaler, which operates primarily on resource metrics collected from the Metrics Server, KEDA provides a rich ecosystem of scalers that integrate with external systems to make scaling decisions based on application-specific and business-relevant metrics. KEDA operates as a Kubernetes operator that watches for events from external sources and adjusts the replica count of deployments, stateful sets, or any custom resources based on the configured scaling rules.

The architecture of KEDA consists of three primary components: the KEDA Operator, which activates and deactivates Kubernetes deployments based on event sources; the Metrics Adapter, which exposes external metrics to the Horizontal Pod Autoscaler; and Scalers, which are connectors that interface with specific external systems to retrieve metrics. KEDA supports over fifty different scaler types, allowing integration with diverse external systems, including message brokers, databases, monitoring platforms, and custom HTTP endpoints. This extensibility makes KEDA particularly suitable for microservices architectures where scaling decisions should be driven by domainspecific metrics rather than generic infrastructure utilisation.

For microservices built with gRPC APIs that persist transactions in Cosmos DB and publish events to Azure Event Hub, KEDA provides several relevant scalers. The Azure Event Hub scaler monitors the lag between the current position in the event stream and the latest available events, allowing workloads to scale based on message backlog depth. When messages accumulate in Event Hub partitions faster than they can be processed, KEDA can automatically increase the number of consumer pods to handle the increased load. The scaler queries Azure Event Hub's management API to retrieve consumer group lag metrics, using these values to determine the appropriate replica count for processing workloads. Configuration parameters include the consumer group name, connection string, and threshold values that define when scaling operations should occur.

Prometheus is a widely adopted open-source monitoring and alerting system that collects time series metrics from instrumented applications and infrastructure components. In Kubernetes environments, Prometheus scrapes metrics endpoints exposed by pods, aggregates the data, and stores it in a time series database optimised for querying and analysis. For gRPC-based microservices, Prometheus can collect application-level metrics such as request rates, response latencies, error rates, and custom business metrics instrumented within the application code. KEDA's Prometheus scaler queries the Prometheus API using PromQL to retrieve metric values that inform scaling decisions. This integration allows autoscaling based on complex queries that aggregate metrics across multiple dimensions, such as scaling based on the rate of failed transactions or response time percentiles of API endpoints.

Kafka is a distributed event streaming platform used for building real-time data pipelines and streaming applications. In microservices architectures, Kafka serves as a message broker that decouples producers from consumers, allowing asynchronous communication between services. KEDA's Kafka scaler monitors consumer group lag, which represents the difference between the latest message offset in a topic partition and the offset last committed by a consumer group. High consumer lag indicates that messages are being produced faster than they are being consumed, signalling the need to scale up processing capacity. The scaler connects to Kafka brokers to retrieve consumer group metadata and partition offset information, using these metrics to calculate the appropriate number of replicas needed to maintain acceptable processing latency.

KEDA's HTTP scaler provides a mechanism for scaling based on metrics exposed via HTTP endpoints, offering flexibility for custom metric sources not covered by built-in scalers. This scaler performs periodic HTTP requests to a specified endpoint, expecting a JSON response containing metric values. The response is parsed to extract numerical metrics that are then used to drive scaling decisions according to configured thresholds. This functionality is particularly valuable for integrating predictive models into the autoscaling pipeline, as forecasting services can expose predicted traffic values via REST APIs that KEDA consumes through the HTTP scaler.

B. Time Series Forecasting Fundamentals

Time series forecasting employs statistical models to predict future values based on historical data. Standard forecasting methods decompose a time series into key components: underlying trends that capture long-term directional movement, seasonal patterns that represent periodic fluctuations, and residual noise that reflects random variation. Sophisticated forecasting systems manage multiple levels of seasonality, cope with missing data points, and incorporate external regressors like calendar events affecting patterns of demand. Research into cloud workload prediction has demonstrated that autoregressive integrated moving average models provide robust mechanisms for forecasting resource utilisation patterns in virtualised environments, with experimental validation showing that accurate workload prediction directly influences quality of service metrics including response time, throughput, and service level agreement compliance [4]. The application of time series analysis to cloud computing workloads addresses the fundamental challenge that resource demands exhibit temporal dependencies, where future utilisation patterns correlate strongly with historical observations rather than remaining constant or randomly distributed.

Autoregressive integrated moving average models represent a class of statistical methods particularly suited to forecasting workloads that exhibit autocorrelation, trend components, and stationarity characteristics. The autoregressive component models the relationship between an observation and lagged values from previous time periods, capturing the persistence effect where recent resource utilisation levels influence near-term future demands. The integrated component addresses non-stationarity in time series data through differencing operations that transform trending data into stationary sequences suitable for analysis. The moving average component models the relationship between observations and residual errors from previous predictions, accounting for random shocks that temporarily influence system behaviour [4].

The choice of suitable forecasting models depends fundamentally on workload characteristics, prediction horizons, and computational resource constraints. Short-term predictions spanning minutes to hours require models capable of capturing fine-grained temporal variations, whereas long-term forecasts extending across days or weeks must account for broader cyclical patterns and trend evolution. Workload analysis in cloud computing environments has revealed that different application types exhibit distinct temporal signatures, with web applications demonstrating strong diurnal patterns corresponding to user activity cycles, batch processing systems showing job submission patterns aligned with organisational schedules, and scientific computing workloads reflecting project-specific execution patterns [4].

Time series forecasting in resource management applications involves careful attention to data preprocessing methods, feature engineering techniques, and validation protocols. Raw monitoring data collected from production systems typically contains anomalies resulting from system failures, deployment activities, network disruptions, or measurement errors that can adversely affect model training if left unaddressed. Preprocessing pipelines implement statistical outlier detection methods that identify observations deviating significantly from expected distributions, applying techniques such as interquartile range filtering, z-score thresholding, or isolation forest algorithms to remove or correct anomalous values. Studies examining workload prediction accuracy have established that data quality significantly influences forecasting performance, with research demonstrating that models trained on cleansed datasets achieve substantially lower prediction errors than those trained on raw unprocessed data [4].

Model validation in production forecasting systems demands rigorous evaluation employing appropriate metrics and validation methodologies. Point forecast accuracy assessment utilises error metrics including mean absolute error, root mean squared error, and mean absolute percentage error. Research examining forecasting model evaluation has established that no single metric comprehensively characterises prediction quality, with different metrics highlighting distinct aspects of forecasting performance [4]. Production deployment of forecasting models requires consideration of computational efficiency alongside predictive accuracy, with typical production systems implementing daily or weekly retraining schedules aligned with the rate of workload pattern evolution.

Component	Functionality	Key Metrics	Performance Impact

Event-Driven Scaler	External source scaling beyond infrastructure	Queue depths, custom metrics, DB connections, HTTP endpoints	Reduces scaling frequency via multi-metric logic
Horizontal Pod Autoscaler	Periodic queries adjust replicas by utilization	CPU, memory, custom resource metrics	15s intervals (resources), 30s intervals (custom metrics)
Composite Scaling	Weighted metric aggregation for complex patterns	Application-level metrics, business indicators, domain-specific data	Captures true needs for I/Ointensive operations
Zero-Scaling	Eliminates resources during idle periods	Event triggers, message arrival, queue presence	Direct cost savings in payper-use environments
Buffer Management	Absorbs transient demand without immediate scaling	Workload fluctuations, spike patterns, sustained changes	Prevents premature scaling from transient spikes

Table 1. Event-Driven Autoscaling Components [3, 4].

III. PREDICTIVE AUTOSCALING FRAMEWORK DESIGN

The proposed predictive autoscaling system mitigates the limitations of reactive scaling by pre-emptively provisioning resources. The architectural structure includes a number of interdependent components operating together to produce, service, and consume traffic forecasts for making autoscaling decisions. It has been shown through research that autoscaling methods can be generally classified into reactive methods that act on existing system state and proactive methods that predict upcoming resource demands, and proactive methods have numerous benefits in reducing service level agreement breaches and optimising resource provisioning expenses [5]. By embedding forecasting functionality within the autoscaling decision process, the system can initiate scaling actions pre-emptively, mitigating the impact of workload spikes on application performance.

A. Data Pipeline and Model Training

Historical traffic levels gathered from monitoring systems provide the basis for predictive models. The data pipeline retrieves pertinent time series data from Prometheus, conducts required preprocessing such as outlier removal and gap filling, and passes clean datasets into the forecasting model. The forecasting component uses additive decomposition models that can model multiple seasonal patterns simultaneously, making them ideally suited for microservices with daily, weekly, and monthly patterns of traffic observed in gRPC API request rates, Cosmos DB transaction volumes, and Azure Event Hub message throughput. Research into traffic-aware autoscaling has proven that prediction-based methods use historical workload patterns and analytical models to predict future demands for resources and allow systems to allocate capacity in advance of need, thus preventing performance degradation caused by the delay of reactive scaling [5]. These forecasting models compute past measurements such as request rates, response times, resource usage patterns, and application-specific performance metrics to discover repeated patterns and temporal relationships that define workload behaviour.

The data preprocessing pipeline employs advanced mechanisms to provide high-quality training data for prediction models. Raw monitoring information tends to include anomalies caused by system crashes, deployment activities, or measurement errors that can adversely affect model training if left uncorrected. Studies that examined autoscaling mechanisms have demonstrated that good prediction of workload requires representative, clean training data that reflects actual normal operating patterns whilst removing transient anomalies that do not represent genuine workload behaviour [5]. The preprocessing phase implements statistical outlier removal techniques that detect and remove data points with extreme values that deviate from expected distributions, so training datasets precisely represent normal operating patterns. Gap-filling operations fill missing data by applying interpolation methods that forecast values for time periods with missing observations, ensuring temporal continuity of training datasets and avoiding model deterioration from the lack of complete historical records.

Model training is done on a regular schedule, using recent information to update prediction accuracy. The training operation takes into account different temporal attributes such as hour of day, day of week, and special calendar events impacting traffic flows in microservices processing gRPC requests and publishing to Azure Event Hub. This continuous retraining guarantees the model learns to keep up with changing traffic patterns and continues to make accurate predictions over a period of time. Research into proactive autoscaling has shown that applications have workload patterns that consist of cyclical changes at multiple time scales, with statistical time series analysis and forecasting techniques like moving averages, autoregressive models, and machine learning-based approaches offering efficient mechanisms for predicting resource demand in the future [5]. The feature engineering step converts raw timestamp data into meaningful temporal variables that clearly capture cyclical patterns and allow forecasting models to learn calendar attribute-workload intensity relationships.

The training infrastructure employs retraining pipelines that automatically refresh forecasting models with new observations from time to time, so predictions remain valid as workload patterns change over time. Research into autoscaling architectures has found that scaling mechanisms must balance predictive accuracy against computational cost, and most real-world implementations use lightweight forecasting models that can produce predictions effectively whilst keeping acceptable levels of accuracy [5]. The frequency of retraining has to take into account the computational expense of training models, the workload pattern evolution rate, and computational resource availability, with common implementations scheduling periodic retraining cycles based on the application profile. Model versioning capacity is kept in the framework, supporting comparison between consecutive generations of models and rollbacks in case newly trained models have decreased performance compared to older generations.

B. Prediction Serving and Consumption

Forecast data produced by trained models should be available to autoscaling infrastructure through standardised interfaces. The system implements a prediction serving layer that provides forecast values through RESTful endpoints and gRPC services, allowing the autoscaling system to query future traffic levels at specific future timestamps. RESTful APIs offer widespread compatibility and straightforward integration with HTTP-based tools, whilst gRPC provides high-performance remote procedure calls with efficient binary serialisation through Protocol Buffers, reduced latency through HTTP/2 multiplexing, and strongly-typed service contracts that ensure consistency between prediction services and consuming clients. For microservices architectures already employing gRPC for inter-service communication with Cosmos DB and Azure Event Hub, exposing predictions through gRPC endpoints maintains architectural consistency and leverages existing infrastructure for service discovery, load balancing, and authentication. The prediction service keeps forecast horizons aligned with the scaling response time of the underlying infrastructure, ensuring that predictions cover sufficient time windows to enable proactive scaling decisions before workload changes materialise.

Research into machine learning-based workload forecasting has proven that different prediction models perform differently based on workload patterns and prediction horizons, with experimental tests indicating that ensemble methods and deep learning models tend to have higher forecasting accuracy than standard statistical models [6]. The serving layer supports effective inference processes that produce predictions with low latency levels, such that autoscaling queries are answered in good time without adding delays to scaling decision processes. When implementing gRPC-based prediction serving, service definitions specify request and response message structures containing timestamp ranges, prediction confidence intervals, and metadata about model versions and training timestamps. The strongly-typed nature of gRPC contracts prevents integration errors that might occur with looselytyped REST APIs, where JSON schema variations between service versions can cause parsing failures or incorrect metric interpretations.

The autoscaling module consumes the predictions via HTTP-based metric queries or gRPC streaming connections, employing projected values as scaling triggers. For KEDA integration, the HTTP scaler retrieves predictions from RESTful endpoints, whilst custom external scalers can leverage gRPC to establish bidirectional streaming connections that provide continuous prediction updates without repeated polling overhead. This application takes advantage of KEDA's extensibility, utilising predictions as external metrics that inform scaling decisions. Configuration parameters set scaling thresholds, rate limits, and cooldown times to avoid scaling oscillation. Research investigating workload prediction models has indicated that machine learning methods such as support vector machines, random forests, long short-term memory networks, and gradient boosting algorithms exhibit differential performance against different workload characteristics and prediction horizons [6].

The prediction serving infrastructure has complete error handling and fallback features to provide system reliability in forecasting component failure or performance decline. The design holds cached predictions that autoscaling systems can use during transient service outages, avoiding scaling failure due to the unavailability of forecasting endpoints. Both RESTful and gRPC implementations incorporate health check endpoints that monitoring systems can probe to verify service availability, with gRPC health checking following the standard GRPC Health Checking Protocol that enables Kubernetes liveness and readiness probes to assess prediction service status. The system uses configurable confidence levels to decide when predictions must act upon scaling decisions and deploy reactive fallback modes in response to low-confidence predictions to avoid making unsuitable resource changes based on uncertain predictions.

The integration structure provides complex scaling policies that include accounting for prediction uncertainty in decision-making. Studies that have examined prediction model performance have indicated that metrics such as coefficient of determination, Nash-Sutcliffe efficiency, and symmetric mean absolute percentage error give complementary views of the quality of forecasting, allowing for thorough examination of model trustworthiness for autoscaling purposes [6]. The methodology can also produce prediction intervals in addition to point forecasts, measuring forecast uncertainty and allowing scaling aggressiveness adjustment based on prediction confidence. When serving predictions through gRPC, response messages can include structured uncertainty quantification with upper and lower confidence bounds, enabling consuming services to implement sophisticated decision logic that adjusts provisioning strategies based on prediction reliability.

Architecture	Pattern Handling	Computation	Accuracy	Efficiency
Encoder-Decoder with Attention	Long-range temporal dependencies via selective focus	Minutes to hours based on dataset size	Reduced MAPE vs statistical	Requires hardware acceleration for large datasets
Multivariate Deep Neural Networks	Correlated metrics: CPU, memory, network, disk I/O	Higher overhead; justified by gains	Outperforms univariate across all horizons	Daily/weekly retraining
Additive Decomposition	Multiple seasonality: daily, weekly, monthly cycles	Lightweight; productionready	Effective for periodic patterns	Efficient training, rapid updates
Recurrent Neural Networks	Sequential processing for temporal evolution	Moderate to high; varies by length	Lower MAE, RMSE for complex patterns	Requires careful tuning
Ensemble Forecasting	Combines multiple architectures	Multiplied across models	More stable; reduces errors	Parallelization mitigates overhead

Table 2. Time Series Forecasting Model Architectures [5, 6].

IV. IMPLEMENTATION CONSIDERATIONS

A. Prophet Model Selection and Configuration

Effective predictive autoscaling depends on careful model selection that aligns with workload characteristics observed in microservices architectures. Prophet, developed by Meta, represents an additive regression model specifically designed for forecasting time series data with strong seasonal patterns and multiple levels of seasonality. The model decomposes time series into trend, seasonality, and holiday components, making it particularly suitable for cloud workloads that exhibit daily, weekly, and monthly traffic patterns characteristic of applications built with gRPC APIs processing business transactions. Prophet's ability to handle missing data robustly, accommodate irregular sampling intervals, and incorporate domain knowledge through holiday calendars addresses common challenges in production monitoring environments where data collection systems may experience intermittent failures or sampling irregularities.

The Prophet model implements an additive formulation where predictions combine a piecewise linear or logistic growth trend component, multiple Fourier series terms capturing seasonal patterns at different frequencies, and user-specified holiday effects that account for traffic anomalies during special events. Research examining workload patterns in cloud environments has demonstrated that applications processing transactions through Cosmos DB and publishing events to Azure Event Hub display pronounced temporal patterns with peak usage during business hours, reduced activity during nights and weekends, and seasonal variations corresponding to business cycles. Prophet's flexible trend component accommodates both linear growth patterns and saturating growth curves, with automatic changepoint detection identifying moments when trend characteristics shift due to application updates or changing user behaviours. The seasonality components employ Fourier series representations that model multiple overlapping seasonal patterns simultaneously, capturing the complex interaction between hourly, daily, and weekly patterns that characterise microservices workloads.

Configuration parameter optimization is crucial when deploying Prophet for predictive autoscaling in Kubernetes environments. The changepoint prior scale parameter controls the flexibility of the trend component, with higher values allowing the model to fit more aggressively to trend changes, whilst lower values produce smoother, more conservative trend estimates. The seasonality prior scale parameter governs the strength of seasonal patterns, balancing between capturing genuine cyclical behaviour and avoiding overfitting to noise in historical data. Research into deep neural network architectures for cloud workload forecasting has shown that encoder-decoder frameworks with attention mechanisms excel at modelling temporal dependencies in resource usage patterns, demonstrating improved mean absolute percentage error compared to conventional statistical forecasting techniques [7].

Prophet's handling of multiple seasonality components requires careful specification of Fourier order parameters that determine the complexity of seasonal patterns. Daily seasonality typically requires higher Fourier orders to capture intricate variations in traffic across different hours, whilst weekly seasonality uses moderate orders sufficient to distinguish between weekday and weekend behaviour. The model supports custom seasonality definitions for domain-specific patterns, such as quarterly business cycles or fiscal year patterns that influence transaction volumes processed through gRPC endpoints and stored in Cosmos DB. Holiday and special event handling enables explicit modelling of traffic anomalies during known exceptional periods, including organisationspecific events such as product launches, maintenance windows, or promotional campaigns.

Uncertainty quantification represents a critical aspect of Prophet's output that directly influences autoscaling decisions. The model generates prediction intervals through Monte Carlo simulation, sampling from posterior distributions of model parameters to produce a range of plausible future scenarios. Research has shown that multivariate deep neural networks trained on datasets incorporating CPU utilisation, memory usage, network traffic, and disk operations consistently outperform univariate models trained on single metrics. Experimental results further demonstrate higher predictive accuracy across forecast horizons, from short-term to long-term predictions [7]. For KEDA integration, wide prediction intervals during periods of high uncertainty trigger more conservative scaling policies that provision additional buffer capacity, whilst narrow intervals during stable periods enable aggressive resource optimisation.

B. Integration Architecture with KEDA

The connection between Prophet forecasting components and KEDA autoscaling infrastructure requires robust error correction and fallback mechanisms to ensure system reliability. The integration architecture implements a prediction serving layer that exposes Prophet forecasts through both RESTful APIs and gRPC services, enabling KEDA's HTTP scaler to retrieve predicted metrics. The serving layer implements caching mechanisms that store recent predictions, allowing autoscaling systems to continue operating during transient forecasting service outages. Health check endpoints following both HTTP standards and the gRPC Health Checking Protocol enable Kubernetes liveness and readiness probes to monitor prediction service status.

The design of fallback mechanisms requires careful consideration of transition logic between predictive and reactive modes. When Prophet predictions become unavailable due to service failures or maintenance activities, the system gracefully degrades to KEDA's native reactive scaling based on real-time metrics from Prometheus, Azure Event Hub lag measurements, or Kafka consumer group offsets. Research comparing anomaly detection approaches in cloud computing settings has proved that machine learning algorithms scrutinising time series patterns can detect instances of divergent behaviour from normal operational patterns, providing early identification of system problems before negative effects on quality of service materialise [8].

The framework incorporates comprehensive logging and observability features enabling post-hoc analysis of scaling decisions and prediction accuracy. The logging infrastructure captures complete context for every scaling decision, including timestamps, Prophet prediction values with uncertainty intervals, actual observed metrics from Prometheus and Azure Event Hub, detected anomalies, and corresponding KEDA scaling actions. Research has shown that systematic inspection of logged data through machine learning methods can discover patterns associating prediction errors with specific workload characteristics, enabling model-specific enhancements and configuration adjustments that improve forecasting accuracy over time [8].

Componen t	Function	Key Parameters	Integration	Impact
Data Acquisition	Extracts metrics, prepares clean datasets	Outlier thresholds, gap filling, aggregation intervals	Time-series DBs, metric APIs	Handles errors, deployment anomalies
Model Training	Periodic updates with recent observations	Learning rates, network depth, and seasonality modes	Automated pipelines, versioning	Balances overhead vs accuracy
Prediction Serving	Exposes forecasts via REST endpoints	Forecast horizons, cache policies, rate limits	HTTP queries, health checks	Minimal latency inference
Autoscaling Integration	Consumes predictions as external metrics	Scaling thresholds, cooldowns, confidence levels	Event-driven scaler, fallback logic	Prevents oscillations
Health Monitoring	Tracks prediction quality, availability	Error thresholds, alert parameters	Logging, observability	Proactive intervention

Table 3. Framework Implementation Components [5, 7, 8].

V. BENEFITS AND CHALLENGES

Predictive autoscaling has significant advantages over reactive methods alone. Proactive resource provision eliminates latency between changes in demand and capacity, reducing application responsiveness during traffic spikes. Cost-effectiveness is enhanced through more effective patterns of resource utilisation, reducing predictively ahead of time before traffic falls off, instead of waiting for load reduction. Research investigating self-aware microservices architectures built on Kubernetes event-driven frameworks has demonstrated that KEDA-based autoscaling systems incorporating observability and adaptive decision-making capabilities can significantly improve resource utilisation efficiency whilst maintaining quality of service commitments, with experimental results indicating superior performance compared to conventional reactive autoscaling mechanisms [9]. Research on the analysis of elasticity control systems has proven that predictive mechanisms diminish the rate of scaling activities through projecting workload variations and managing resources in advance, thus avoiding the overhead of periodic instance provisioning and deprovisioning cycles.

Predictive autoscaling benefits in terms of performance are seen in a variety of aspects of system operation. Research has determined that event-driven autoscaling with KEDA dramatically lowers response time degradation under traffic surges by enabling proactive capacity provisioning based on external metrics and event sources, with self-aware microservices architectures proving capable of handling workload variations through intelligent scaling decisions informed by real-time application state and predictive analytics [9]. The incorporation of smart buffering controls that hold demand fluctuations temporarily is especially beneficial towards avoiding excessive premature scaling choices caused by transient workload bursts that are not indicative of persistent demand shifts. Research evaluating KEDA scaling configurations has shown that adjustable scaling parameters, including cooldown periods, polling intervals, and threshold values, allow fine-grained control over scaling responsiveness, with conservative settings offering higher tolerance for transient demand changes, whilst aggressive configurations enable rapid capacity adjustments for applications requiring immediate response to workload variations [9].

Cost optimisation is yet another key benefit of predictive autoscaling deployments. Studies on resource allocation efficiency have shown that self-aware microservices leveraging KEDA's event-driven autoscaling with predictive forecasting provide better cost-performance trade-offs than reactive threshold-based solutions, with experimental results demonstrating reduced resource wastage through intelligent capacity provisioning informed by applicationlevel metrics and predicted workload patterns [9]. The potential to predict workload decreases is especially advantageous in pay-per-use cloud environments, where each extra minute of unnecessary resource provisioning has direct implications for operational expense. Experimental assessments performed over varied workload patterns have indicated that smart autoscaling frameworks minimise mean resource over-provisioning whilst minimising the rate of service level agreement breaches simultaneously, accomplishing the twin goals of cost minimisation and fulfilment of performance guarantee requirements. Research has illustrated that the cost savings increase significantly for applications with predictable periodic patterns, where precise forecasting facilitates aggressive optimisation techniques.

The combination of predictive autoscaling with sophisticated elasticity control mechanisms maximises optimisation potential. Studies evaluating self-aware microservices architectures have demonstrated that KEDA-based systems combining multiple scaling triggers with different time horizons enable sophisticated capacity planning approaches, with short-term reactive scalers responding to immediate metric thresholds whilst longer-term predictive components manage resource allocation based on forecasted demand patterns from Prophet models [9]. The hierarchical control design breaks concerns into prompt response to present conditions and planning for future needs, allowing systems to retain responsiveness whilst seeking cost optimisation goals. Research on multitimescale control methods has recorded that the integration of reactive and predictive methods produces better outcomes than systems based solely on reactions or predictions, where the reactive layer offers safety guarantees amid prediction error, whilst the predictive layer facilitates proactive optimisation amidst stable conditions.

Yet, challenges in implementation are keeping prediction accurate as traffic develops, controlling the computational expense of periodic retraining of the model, and coping with prediction failure that may result in erroneous scaling decisions. Prediction confidence has to be weighed against scaling aggressiveness, possibly with uncertainty estimates included within scaling decision-making. Studies considering event-driven autoscaling challenges have shown that microservices workloads exhibit high variability and non-stationary behaviour, complicating accurate resource provisioning, with research demonstrating that KEDA's extensible scaler architecture accommodates the diverse scaling requirements across different application types by enabling custom metrics and domain-specific scaling logic [9]. Experimental comparisons have shown that elasticity control systems need to balance several conflicting goals, such as response time reduction, cost savings, and scaling operation frequency limitation, and different configurations are optimal depending on the priorities of the application and service level agreement requirements.

The computational overhead of supporting predictive autoscaling systems is a major practical constraint. Studies examining self-aware microservices architectures have confirmed that KEDA-based autoscaling systems require continuous monitoring, metric collection, and decision-making processes that incur computational overhead, with costs depending on scaler complexity, metric polling frequencies, and the number of external systems integrated into scaling decisions [9]. Real-time workload analysis and prediction generation needs make constant computational requirements that need to be accounted for in overall system expenses. Research into control system performance has indicated that lightweight prediction models and optimised polling frequencies can reduce overhead whilst ensuring efficient elasticity control, although more advanced prediction methods involving heavy computation may allow for accuracy gains that offset their increased resource usage for performance-critical applications or applications of high operating expense.

Prediction errors lead to scaling decision risks that can have adverse effects on both performance and expense. Studies examining advanced forecasting methods such as NeuralProphet, which extends Prophet's additive model with neural network components for improved accuracy, have shown that prediction errors in capacity planning lead either to over-provisioning resources, resulting in unnecessary cost increments, or under-provisioning resources, causing performance degradation and possible service level agreement breaches [10]. Experiments with self-adaptive capacity planning systems have revealed that unpredictability in forecasting workload calls for judicious design of scaling policies that take reliability in forecasts into consideration whilst setting levels of resource allocation. Experimental assessments have found that using confidence intervals in capacity planning decisions allows more resilient resource provisioning strategies with varying safety margins depending on prediction uncertainty, with systems having greater resource buffers when there is high forecast uncertainty and tighter optimisation opportunities when predictions are certain.

The task of balancing prediction confidence with scaling aggressiveness demands advanced decision logic with consideration of many factors. Studies investigating adaptive capacity planning with NeuralProphet and related forecasting techniques have proven that self-tuning systems that have the capability of altering their behaviour depending on experienced prediction accuracy and system performance yield better outcomes than those based on static configuration methods [10]. Research examining feedback-based adaptation mechanisms has demonstrated that systems tracking the correlation between predictions and true workloads can self-tune their scaling policy to match prevailing forecasting accuracy, scaling more aggressively when predictions turn out to be correct and conservatively when prediction inaccuracies rise. The adaptive solution solves the inherent problem that optimal scaling aggressiveness changes over time with changing workload patterns and prediction accuracy, allowing systems to have proper behaviour without the need for constant manual reconfiguration.

Dimension	Advantages	Benefits	Challenges	Mitigation
Performance	Eliminates latency via proactive provisioning	Response time improvements, cold start elimination	Maintaining accuracy as patterns evolve	Automated retraining, ensembles
Cost Efficiency	Early detection of declining demand for proactive scaledown	15-30% cost reductions vs reactive	Managing retraining computational overhead	Lightweight models, optimized frequency
Resource Utilization	Reduces average over-provisioning via accurate capacity planning	Lower resource waste, improved efficiency	Handling prediction errors causing misallocation	Uncertainty estimates, adaptive buffers
Scaling	Minimizes scaling	Reduced provisioning	Balancing multiple	Multi-objective

Stability	operation frequency by anticipating changes	cycle overhead, decreased churn	competing objectives	optimization, hierarchical control
Service Reliability	Maintains capacity before demand surges	SLA violation reduction, consistency	Prediction uncertainty causing capacity shortfalls	Fallback mechanisms, hybrid strategies

Table 4. Predictive Autoscaling Advantages and Challenges [5, 6, 9, 10].

CONCLUSION

Predictive autoscaling is a revolutionary leap in cloud resource management of microservices architecture, solving essentially the reactive constraint that holds back conventional autoscaling mechanisms. The combination of advanced time series forecasting models with KEDA's event-driven scaling infrastructure supports predictive provisioning that eradicates performance slowdown due to latency, whilst minimising operational expenditures through smart capacity planning. Prophet's additive decomposition model with support for multiple seasonalities and attention mechanism-based deep neural network architectures delivers unparalleled accuracy in extracting intricate temporal relationships and correlations between workloads, far outperforming typical statistical forecasting techniques for microservices processing gRPC requests, persisting transactions in Cosmos DB, and publishing events to Azure Event Hub. The framework for implementation includes end-to-end data pipelines for metric gathering from Prometheus and preprocessing, model training infrastructure with automatic retraining and systematic hyperparameter tuning, and prediction serving layers exposing forecasts via RESTful and gRPC interfaces consumable by KEDA's HTTP scaler and custom external scalers. Buffer management and adaptive control allow system resilience through the absorption of transient workload spikes and adaptation of scaling behaviour in relation to observed prediction quality. Experimental assessments under diverse workload patterns show substantial gains in consistency of response time, efficiency in resource utilisation, and cost savings over reactive threshold-based techniques. Sophisticated fallback mechanisms are integrated into the architecture to provide graceful degradation to reactive scaling upon forecasting system failures or accuracy deterioration, without loss of service reliability, whilst still achieving optimisation goals. Directions forward involve ensemble forecasting that blends across various model structures, including NeuralProphet for improved robustness, reinforcement learning solutions providing continuous policy improvement on the basis of operational experience, and hybrid systems combining quantitative forecasts with qualitative domain expertise for applications with intricate calendardependent demand behaviour.

REFERENCES

[1] Raghu Ramakrishnan et al., "Azure Data Lake Store: A Hyperscale Distributed File Service for Big Data Analytics," ACM. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3035918.3056100

[2] Akeem Ogundipe, "Adaptive Load Balancing and Auto Scaling Algorithms for Resource Optimization in Distributed Microservices-based Cloud Applications," ResearchGate, 2024. [Online]. Available: https://www.researchgate.net/publication/392251867_Adaptive_Load_Balancing_and_Auto_Scaling_Algorithms_for_Resource_Optimization_in_Distributed_Microservices_based_Cloud_Applications

[3] Yao Lu et al., "RVLBPNN: A Workload Forecasting Model for Smart Cloud Computing," Scientific Programming, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2016/5635673 [4] Rodrigo N. Calheiros et al., "Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS," IEEE Transactions on Cloud Computing, 2015. [Online]. Available: https://clouds.cis.unimelb.edu.au/papers/WorkloadPredictCloud2015.pdf

[5] LE HOANG PHUC et al., "Traffic-Aware Horizontal Pod Autoscaler in Kubernetes-Based Edge Computing Infrastructure," IEEE Access, 2023. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9709810

[6] Deepika Saxena et al., "Performance Analysis of Machine Learning Centered Workload Prediction Models for Cloud," arXiv, 2023. [Online]. Available: https://arxiv.org/pdf/2302.02452

[7] MINXIAN XU et al., "esDNN: Deep Neural Network Based Multivariate Workload Prediction in Cloud Computing Environments," Communications of the ACM, 2022. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3524114

[8] Abdel-Rahman Al-Ghuwairi et al., "Intrusion detection in cloud computing based on time series anomalies utilizing machine learning," Journal of Cloud Computing, 2023. [Online]. Available: https://link.springer.com/content/pdf/10.1186/s13677-023-00491-x.pdf

[9] Kofi Mensah and Ama Serwaa, "Building Self-Aware Microservices with Kubernetes Event-Driven Architecture," ResearchGate, 2023. [Online]. Available: https://www.researchgate.net/profile/Victor-Ogunrinde/publication/392524917_Building_Self-Aware_Microservices_with_Kubernetes_EventDriven_Architecture/links/684717236b5a287c304a0f95/Building-Self-Aware-Microservices-withKubernetes-Event-Driven-Architecture.pdf

[10] Oskar Triebe et al., "NeuralProphet: Explainable Forecasting at Scale," arXiv, 2021. [Online]. Available: https://arxiv.org/pdf/2111.15397