**4. Methodology**

This study adopts a quantitative, experimental methodology to design and evaluate a predictive autoscaling strategy in Kubernetes based solely on the ARIMA (AutoRegressive Integrated Moving Average) model. The methodology is structured into the following phases:

**4.1 Environment Setup**

A local Kubernetes cluster was initialized using Minikube to simulate a controlled, reproducible environment for containerized application deployment. A sample web application was deployed with a predefined workload generation strategy to produce variable CPU utilization patterns over time. The following tools and configurations were employed:

* **Minikube (vX.X)** with Docker driver.
* **Prometheus** for resource monitoring and metrics collection.
* **Grafana** for real-time visualization and validation.
* **Node Exporter** and **Kube-State-Metrics** for detailed metric granularity.

**4.2 Data Collection**

Time-series data of CPU usage (millicores) was collected at 15-second intervals using Prometheus over a continuous observation window of 7 days. The collected metrics were exported to CSV format and preprocessed to handle:

* Missing values (using linear interpolation).
* Noise reduction via rolling averages (window size = 3).
* Resampling to fixed 1-minute intervals to ensure temporal consistency.

**4.3 ARIMA Model Development**

The ARIMA model was employed as the sole predictive mechanism. The modeling process followed these stages:

* **Stationarity Assessment**: Conducted using Augmented Dickey-Fuller (ADF) test. Non-stationary series were differenced until stationarity was achieved (d parameter).
* **Parameter Identification**: Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots were analyzed to determine suitable values for p (autoregressive order) and q (moving average order).
* **Model Selection**: Multiple ARIMA(p,d,q) configurations were evaluated using Akaike Information Criterion (AIC), and the model with the lowest AIC was selected.
* **Training & Validation**: The dataset was split into training (80%) and testing (20%) sets. Model performance was assessed using Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE).

**4.4 Autoscaling Controller Integration**

A custom autoscaler controller was developed as a Kubernetes sidecar application. This controller:

* Periodically queries the ARIMA model predictions using real-time CPU data streamed from Prometheus.
* Forecasts CPU usage over a short time horizon (next 5 minutes).
* Adjusts the number of application replicas by updating the Kubernetes Deployment object via the Kubernetes API, based on predefined CPU utilization thresholds (e.g., scale-out if forecasted usage > 70%).

**4.5 Experimental Scenarios**

Two sets of experiments were conducted:

1. **Reactive Baseline**: Standard Kubernetes Horizontal Pod Autoscaler (HPA) configured to respond to current CPU utilization (threshold = 60%).
2. **Predictive ARIMA Autoscaler**: Autoscaler driven by ARIMA-based forecasted utilization.

Each scenario was subjected to identical workloads (synthetic load via k6 tool) with controlled step and burst patterns. Key performance indicators (KPIs) measured:

* Response latency under load.
* Pod provisioning delay.
* CPU efficiency (requested vs. used).
* Scaling accuracy (true positives vs. false alarms).

**4.6 Evaluation Metrics**

The effectiveness of the ARIMA-based autoscaler was evaluated using the following metrics:

* **Forecast Accuracy**: RMSE, MAPE over test data.
* **Autoscaling Responsiveness**: Time to scale-out/scale-in following predicted load.
* **Resource Efficiency**: CPU wastage and saturation rate.
* **Service Quality**: 95th percentile response latency.