Proactive ML-Driven Auto-Scaling for Cloud Based Applications

Mohsin Khan 19P-0060
Saad Ali 22P-9208
Munazah 21P-8029

Supervised by: Mr. Usama Musharaf

Introduction

- Focuses on improving auto-scaling in cloud based containerized applications.
- Reactive auto-scaling.
- Proactive auto-scaling.
- Machine learning based approach.
- Predicting whether to scale-up or scale-down.

Problem Statement

Variable workloads in cloud environments lead to delayed, reactive scaling, resulting in inefficient resource use, degraded performance, and poor user experience, especially with fluctuating demand. A proactive solution is needed to anticipate and address resource requirements at both node and pod levels.

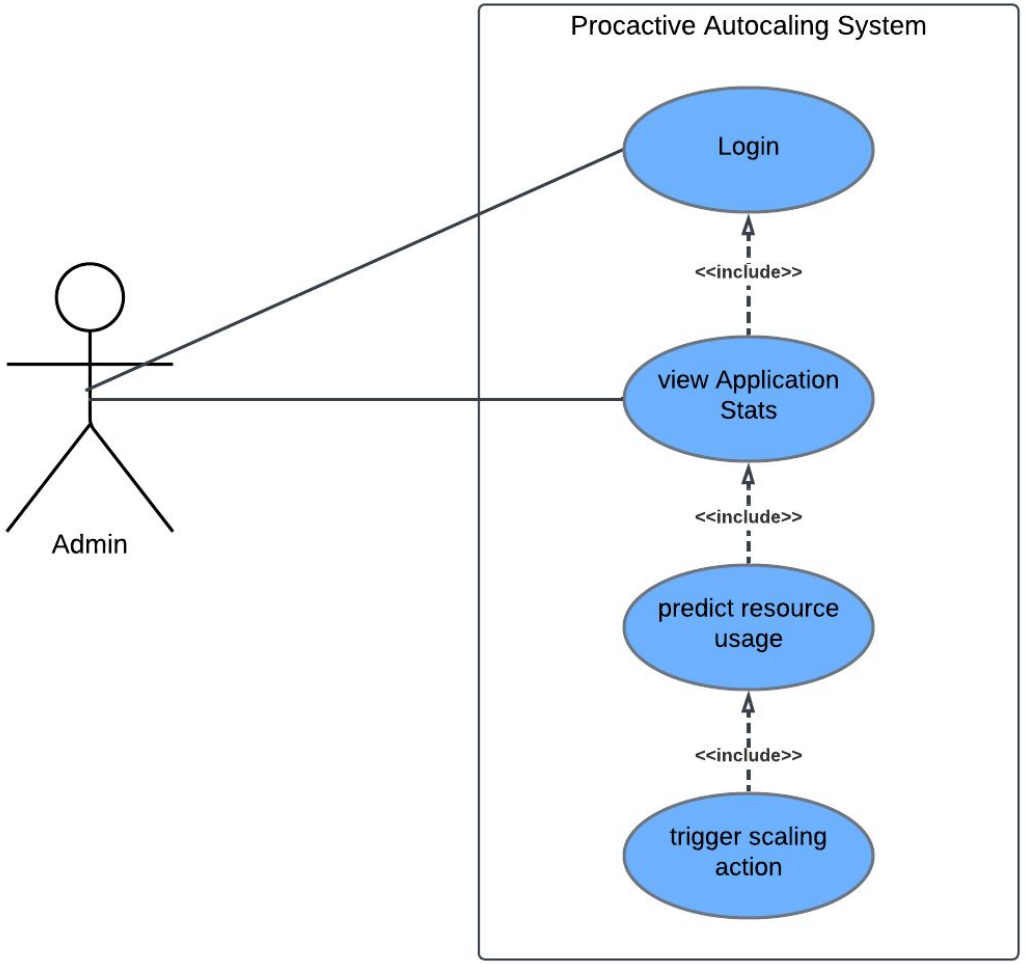
Proposed Solution

This project develops a proactive, machine learning-based auto-scaling system for containerized applications.

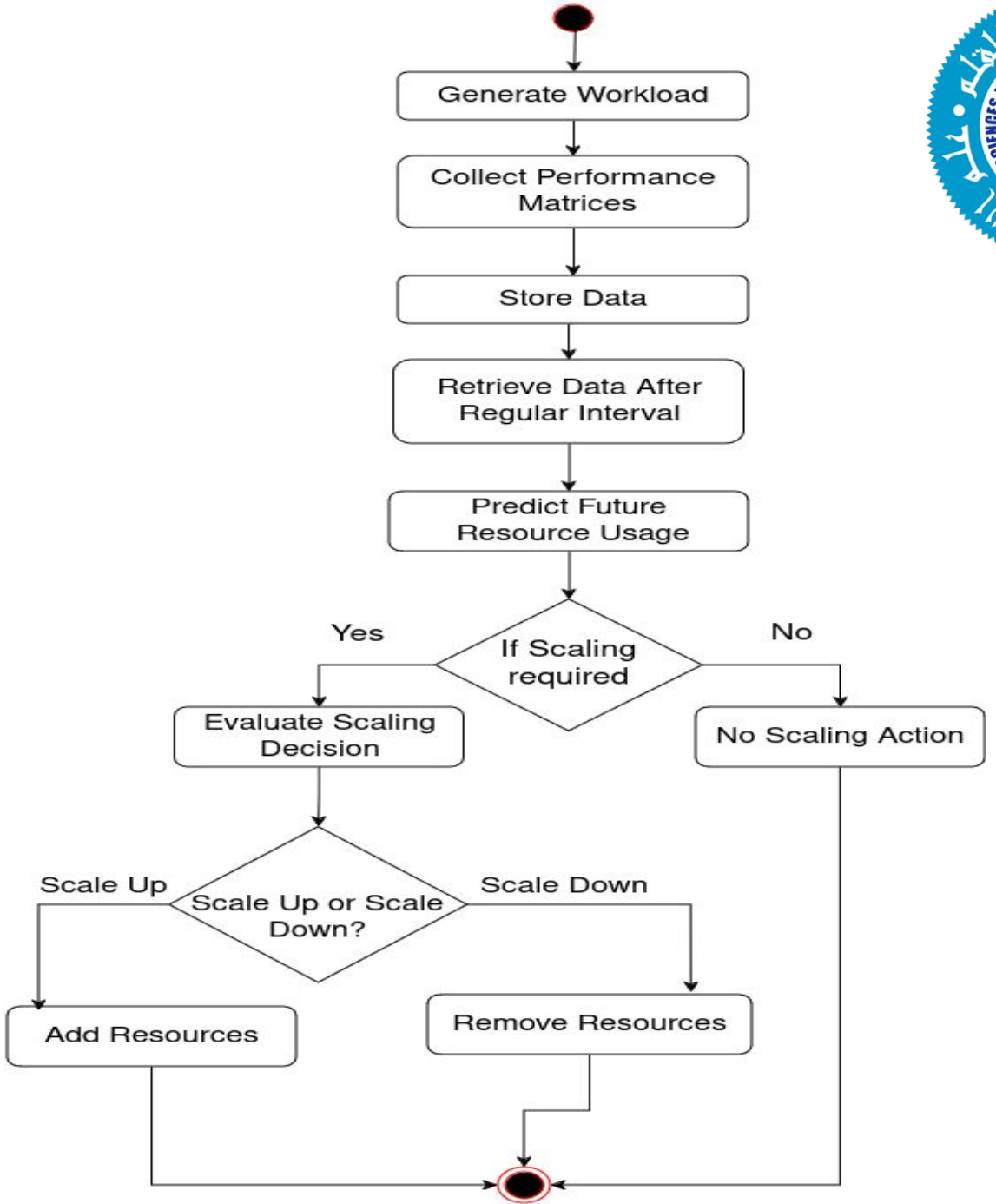
Using the MAPE (Monitor, Analyze, Plan, Execute) framework, it continuously predicts and adjusts resource allocations to optimize performance.

We will train our model on existing data and tested on additional data generated through a custom application, ensuring effective scaling and enhanced user experience under dynamic workloads.

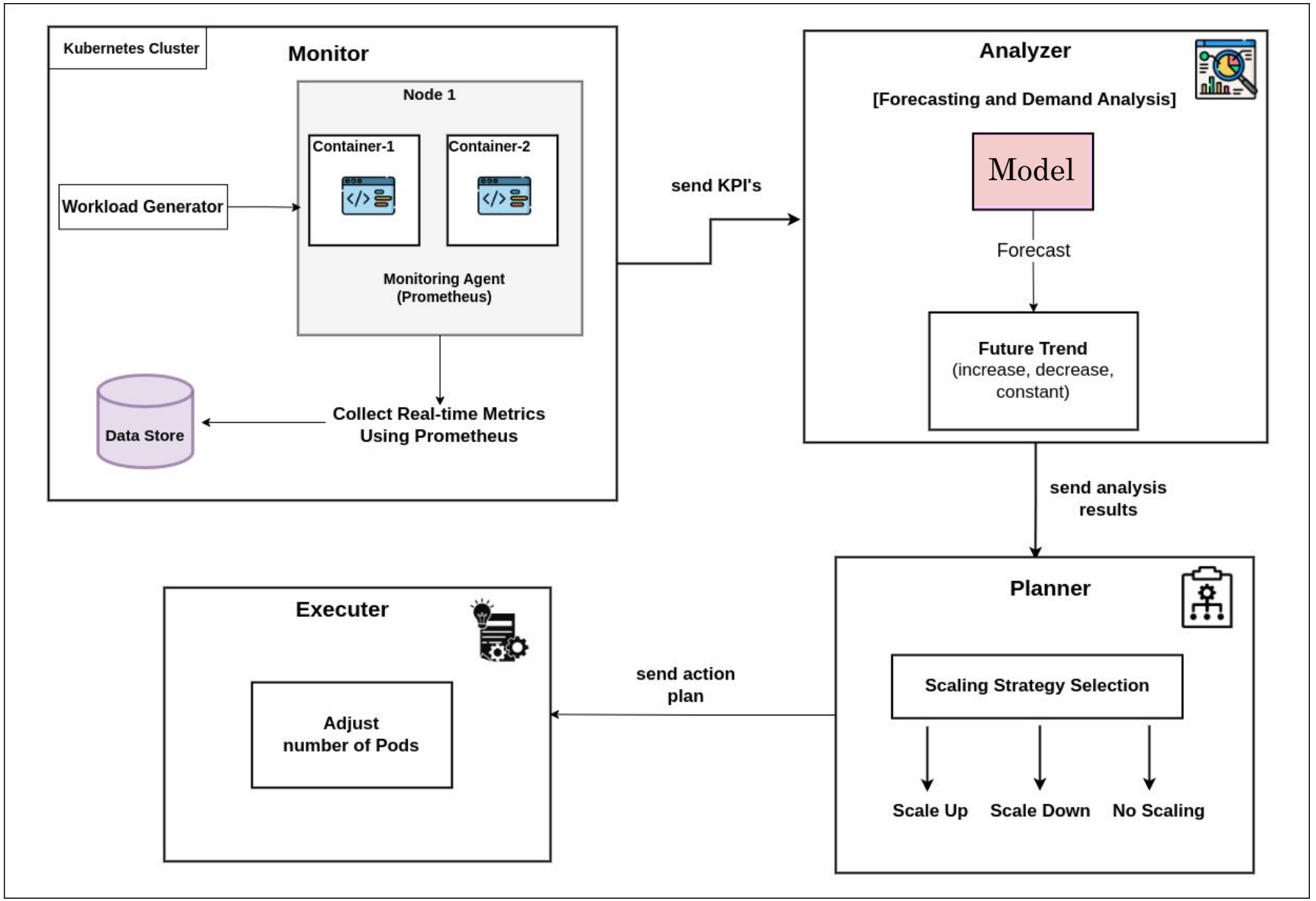
Use Cases/Use Case Diagram



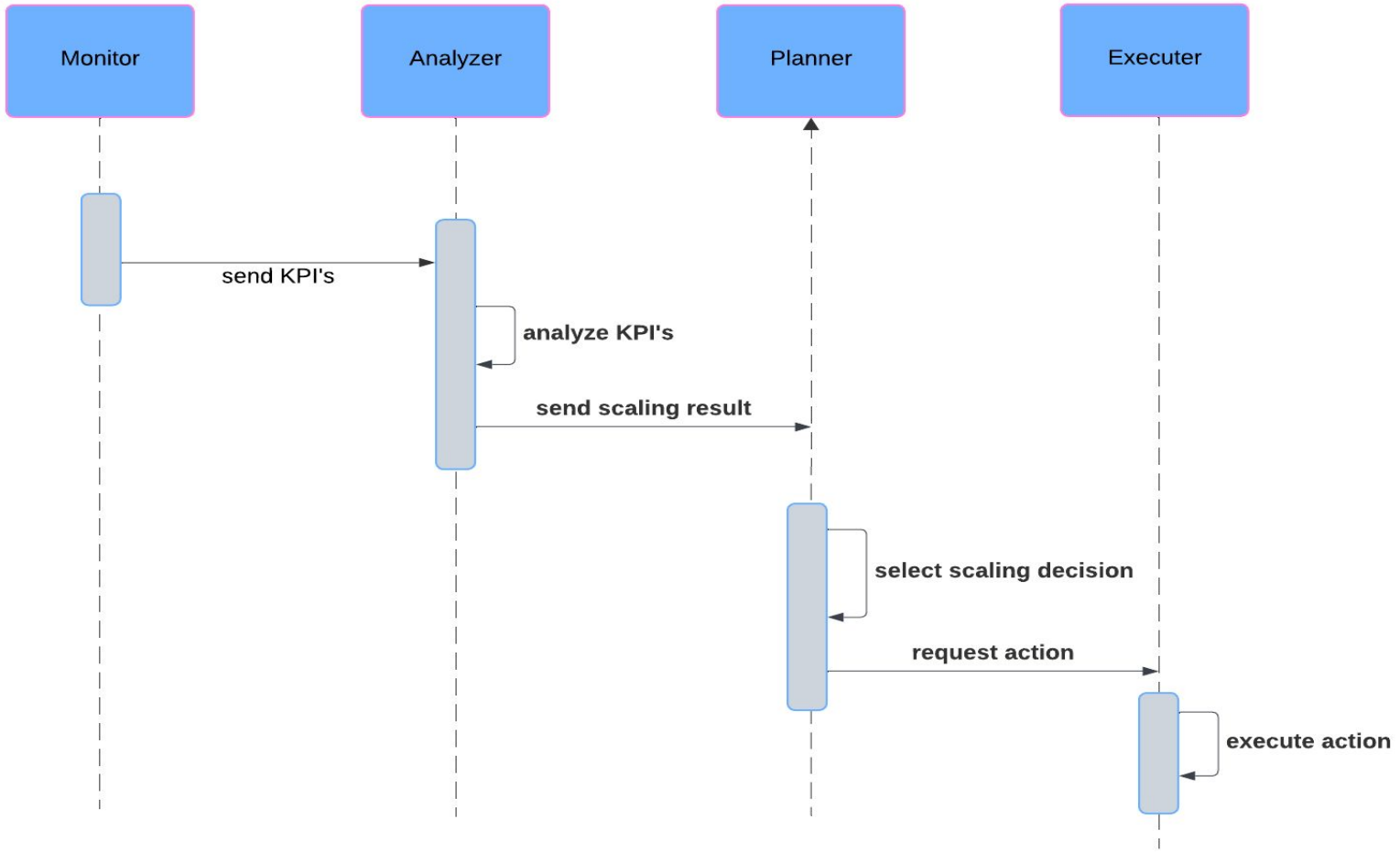
Activity Diagram



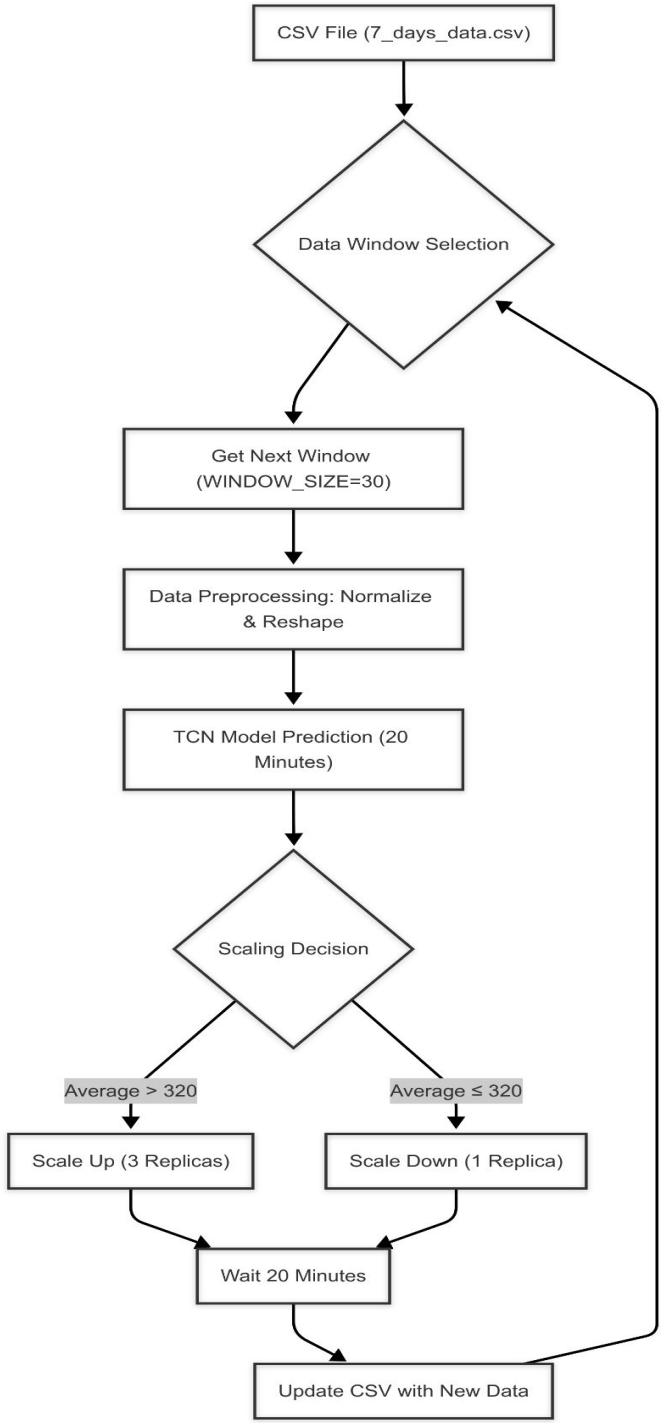
Architecture



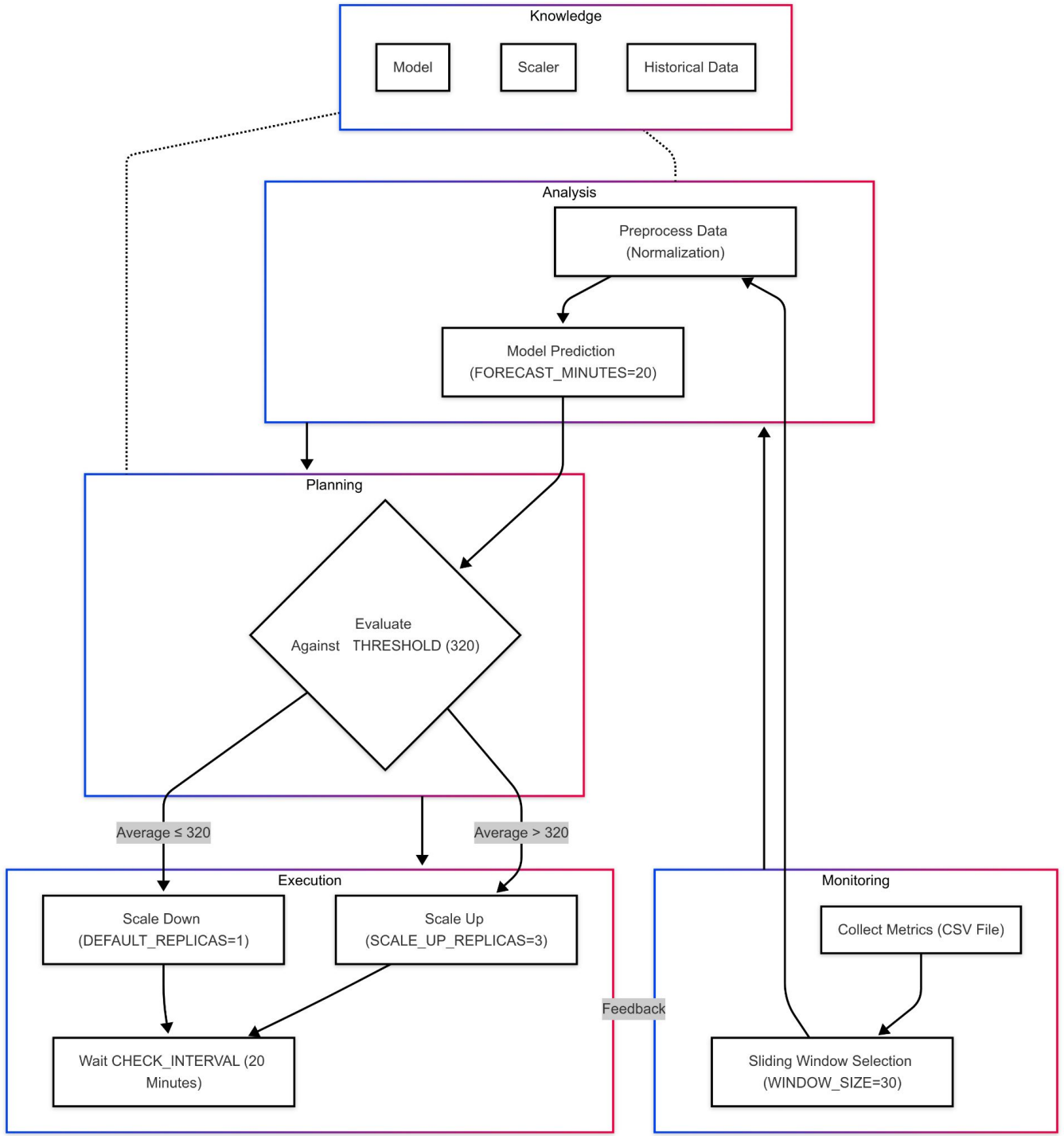
Sequence Diagram



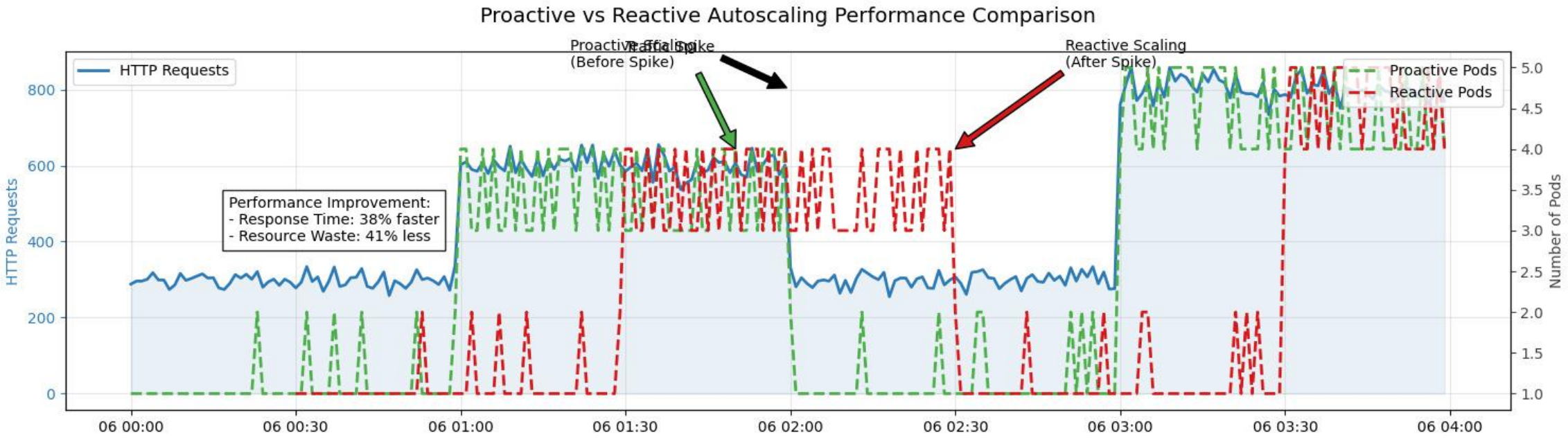
Working


CSV File (7_days_data.csv)
Data Window Selection
Get Next Window (WINDOW_SIZE=30)
Data Preprocessing: Normalize & Reshape
TCN Model Prediction (20 Minutes)
Scaling Decision
Average > 320
Average ≤ 320
Scale Up (3 Replicas)
Scale Down (1 Replica)
Wait 20 Minutes
Update CSV with New Data

MAPE


Knowledge
Model
Scaler
Historical Data
Analysis
Preprocess Data
(Normalization)
Model Prediction
(FORECAST_MINUTES=20)
Planning
Evaluate
Against THRESHOLD (320)
Average ≤ 320
Average > 320
Execution
Scale Down
(DEFAULT_REPLICAS=1)
Scale Up
(SCALE_UP_REPLICAS=3)
Wait CHECK_INTERVAL (20
Minutes)
Feedback
Monitoring
Collect Metrics (CSV File)
Sliding Window Selection
(WINDOW_SIZE=30)

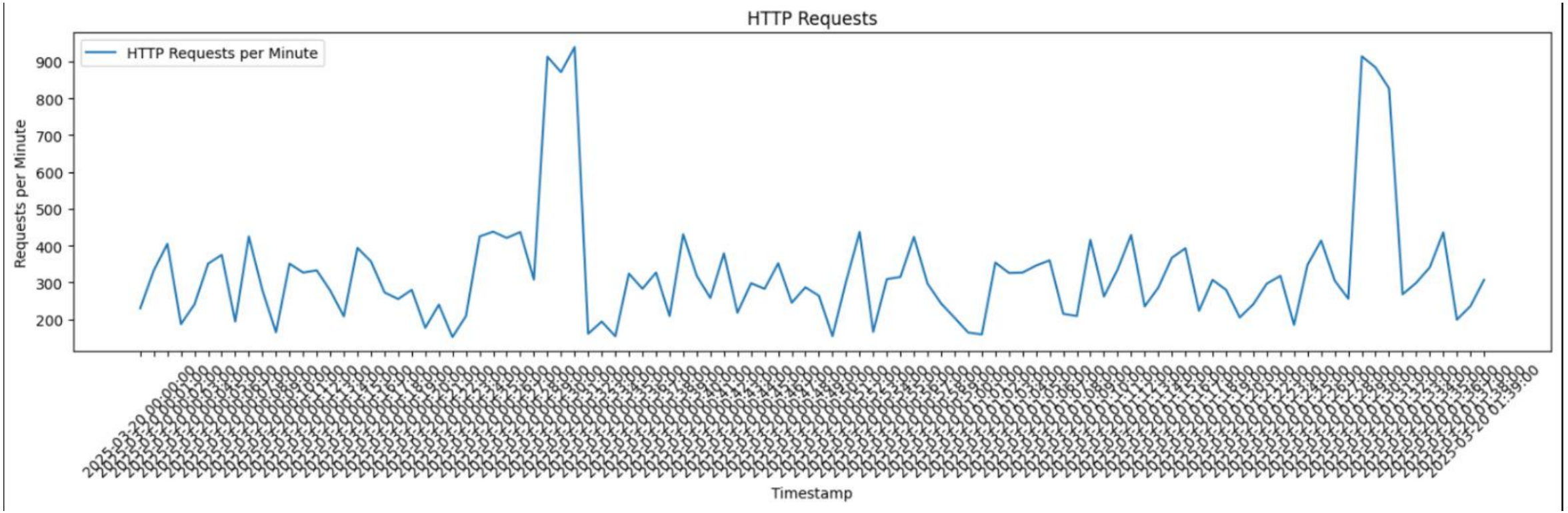
Reactive vs Proactive



Dataset:

- Custom dataset generated from application
- Used a Locust tool to generate load
- Collected metrics data from pods

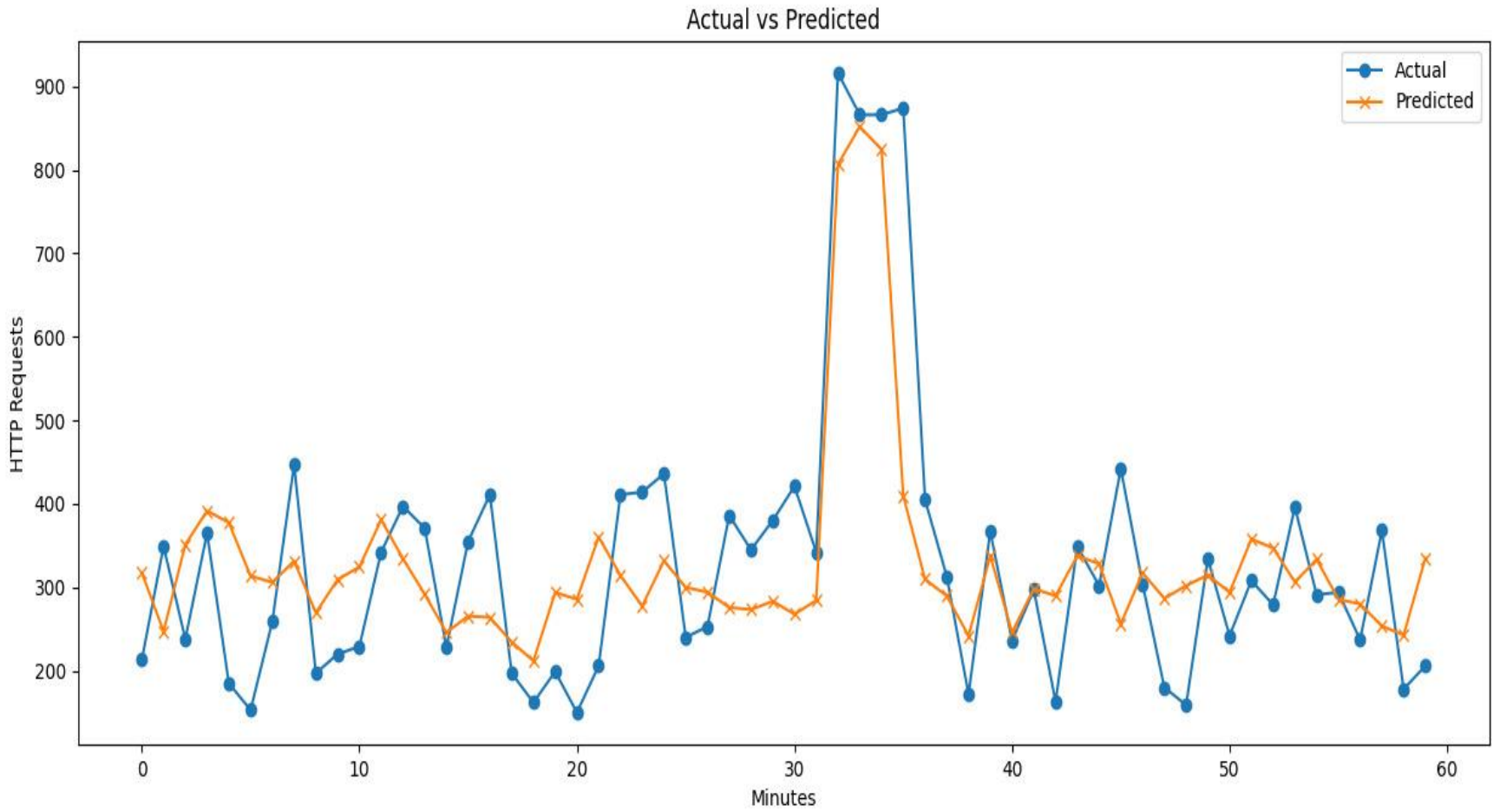
Dataset:


HTTP Requests
HTTP Requests per Minute
Requests per Minute
900
800
700
600
500
400
300
200
Timestamp

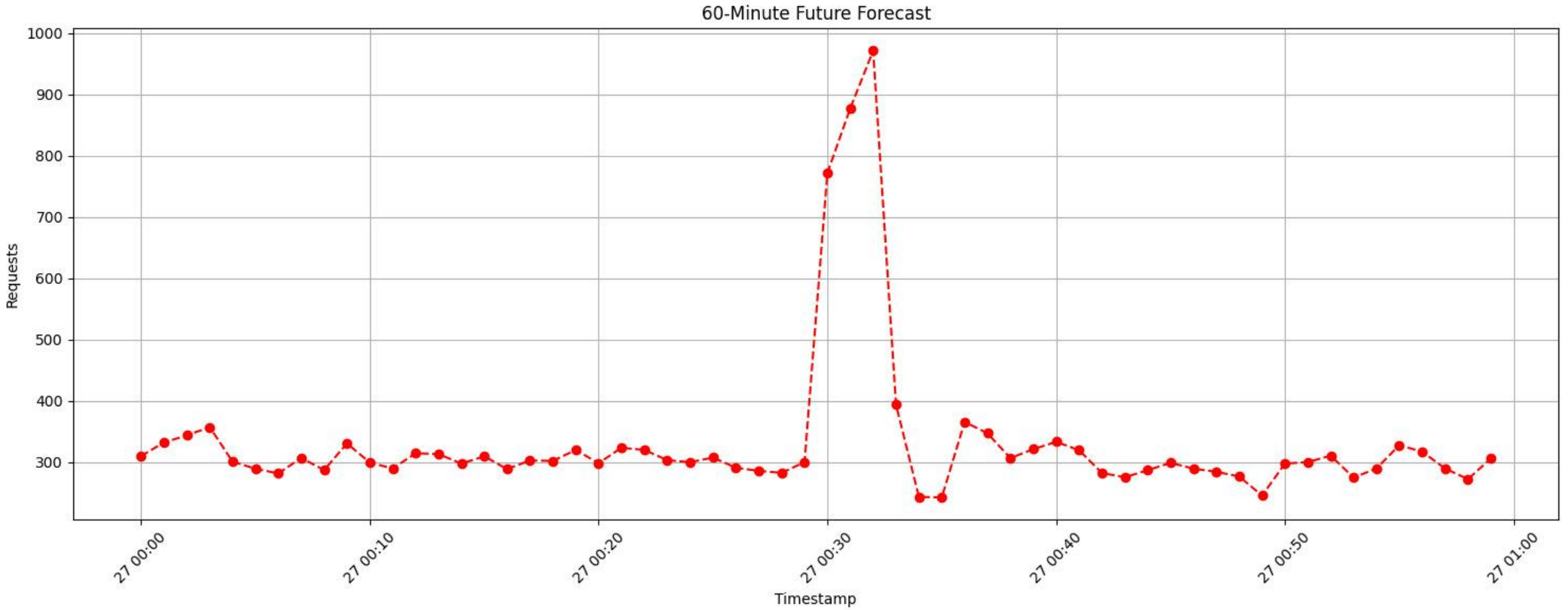
Dataset:


http_requests
1000
500
Trend
400
300
Seasonal
100
0
Resid
500
0
03-20 00
03-20 01
03-20 02
03-20 03
03-20 04
03-20 05

Model Results



Forecasting



Coding Style

- Descriptive Variable Naming(http_requests)
- Modular design
- Dynamically coded
- Logging and monitoring.
- Well-defined code with inline comments
- Utilize built-in functions

Technical Difficulties

- Briding ML Predictions with kubernetes API calls.
- Creating time series data.
- Locust script
- Creating pods
- Access to pods via node ports.
- Resource availability issues

Goals Achieved

- Developed and containerized the application for data generation
- Deployed to kubernetes using Minikube
- Extracted data using prometheus from running pods
- Got predictions using the model
- Scaled pods based on the predicted workload
- Got better scaling than reactive scaling mechanism.

Test Case

ID	Description	Input	Expected Output
TC-01	Take a window from the extracted data	Window length, data store	Extracts the window of desired length
TC-02	Feed data to the model	Input last 30 minutes of request data	60 minutes workload forecast
TC-03	Decide scaling decision based on forecasting	Threshold and average request rate	Successfully scaled
TC-04	Access and extract the minikube pods/services data and display on the dashboard	NodePorts, Minikube IP	Successfully extract and display all the running services data
TC-05	Trigger autoscaler from the dashboard UI	The script to be executed to trigger autoscaler.	Successfully triggered the autoscaler and more replicas were added/removed accordingly

Application of Test Cases

- Tests were applied using the custom built microservices.
- The data was extracted from pods by applying the http requests workload using Locust.
- Different thresholds were tried to make the scaling decision based on the future workload prediction by the model.
- Multiple test rounds were conducted

Testing Outcomes

- System proactively scaled the application up/down before threshold breaches.
- Resource wastage decreased due to prior predictions.
- Performance improved as the resources were added/removed before the bottleneck occurs.

Timeline

Presentations	jan 22- 19 March	March 20 - April 23
Mid Presentation: **1:**Applicaton Deployment **2**: Generate Dataset **3:** Build and Evaluate Model		
Final Presentation : 1: Model Integration 2: Build Dashboard		

Team Work

Moshin Khan

- Research
- Dataset
- Design
- Data Generation
- Model Building
- Documentation
- Presentation

Munazah

- Research
- Dataset
- Design
- Model Building
- Dashboard
- Documentation
- Presentation

Saad Ali

- Research
- Dataset
- Design
- Application development for load extraction
- Dashboard
- Documentation
- Presentation

Thank You