Use Case: AI & ML Agent for Predicting and Remediating Kubernetes Cluster Issues

Team Name : RaKri
Team Members : YUVA SRI B, VIDYA B, SRUTHI B

Table of Contents :

1. Introduction

Kubernetes is a powerful container orchestration platform used to manage applications at scale. However, clusters often experience **failures such as pod crashes, resource exhaustion, and network issues**, leading to service disruptions. To enhance cluster reliability, this project aims to build a **machine learning model** that predicts potential failures before they occur.

By leveraging **historical and real-time cluster metrics**, the model will identify anomalies and forecast system failures, enabling proactive issue resolution. This predictive approach will help reduce downtime, optimize resource utilization, and improve overall system stability.

2. Problem Statement

Kubernetes is widely used for managing containerized applications, providing scalability, automation, and high availability. However, despite its robustness, **Kubernetes clusters are prone to failures** that can impact application performance and availability.

Some of the **common failure scenarios** in Kubernetes clusters include:

- **Pod Failures:** Pods may enter a **CrashLoopBackOff** state due to application crashes, misconfigurations, or resource exhaustion.
- **Resource Bottlenecks:** Excessive **CPU, memory, or disk usage** can degrade node performance, leading to pod evictions or failures.

- **Network Issues: High network traffic, packet loss, and timeouts** can disrupt communication between pods and services.
- **Kubernetes System Errors:** Events such as **node pressure conditions, pod eviction errors** can lead to instability and application downtime.

Predicting such failures **before they occur** is essential for ensuring cluster reliability and minimizing downtime.

The challenge in **Phase 1** of this project is to develop an **AI/ML-based predictive model** that can **analyze historical and real-time Kubernetes cluster metrics** to forecast potential failures.

3. Data Collection

To ensure the accuracy and effectiveness of the model, a **real-time dataset** was collected using **Kubernetes** and **Minikube**. The dataset captures critical cluster metrics that influence system health and failure patterns.

3.1 Tools & Technologies Used

- **Kubernetes**: Orchestration platform for containerized applications
- **Minikube**: Lightweight Kubernetes cluster for local development
- **kubectl**: CLI tool for interacting with the Kubernetes API
- **Python & Pandas**: For data processing and analysis

3.2 Data Collection Procedure

1. **Cluster Setup**
    - A Minikube cluster was deployed on a local machine.
    - Multiple nodes and pods were created to simulate real-world Kubernetes workloads.
2. **Metric Selection**
    - The following cluster metrics were identified as key indicators of failures:
        - **CPU Usage (%)**: Measures node-level CPU consumption
        - **Memory Usage (%)**: Monitors RAM utilization per node
        - **Disk Usage (%)**: Tracks read/write operations and storage pressure
        - **Network Usage**: Identifies traffic levels (Normal, High Traffic)
        - **Pod Status**: Logs pod states (Running, CrashLoopBackOff, Pending)
        - **Kubernetes Event Logs**: Captures warnings/errors such as pod eviction events
        - **System Logs**: Detects node-level issues like kubelet restarts

- **Network Errors**: Flags issues like timeouts or connectivity failures

3. **Data Extraction**
   - Metrics were extracted using the **kubectl get nodes, kubectl logs, and kubectl top pods** commands.
   - The collected logs and metrics were stored in structured CSV/JSON format.
4. **Data Preprocessing**
   - Missing values were handled appropriately.
   - Data was normalized and formatted for ML model training.

3.3 Sample Data Overview

The collected dataset includes timestamps, node-level statistics, and failure indicators. Below is an example of the dataset structure:

Timestamp	Node	CPU_Usage	Memory_Usage	Disk_Usage	Network_Usage	Pod_Status	K8s_Event_Log	System_Log	Network_Error
2025-03-22 13:2	node-1	88%	74%	19%	High Traffic	Pending	Pod eviction error	Disk Failure	Connection Refused
2025-03-22 13:5	node-2	6%	68%	31%	Normal	Unknown	Node NotReady	Kubelet restart detected	Timeout detected
2025-03-22 13:1	node-3	45%	21%	99%	Normal	Failed	Pod eviction error	Kernel Panic	Connection Refused
2025-03-22 13:1	node-4	54%	91%	19%	High Traffic	Running	No Issues	Normal Operation	No Issues
2025-03-22 13:0	node-5	64%	46%	17%	Normal	Pending	Pod eviction error	Kubelet restart detected	Connection Refused
2025-03-22 13:0	node-6	23%	30%	78%	Congested	Running	No Issues	Normal Operation	No Issues
2025-03-22 13:4	node-7	62%	70%	18%	High Traffic	Running	No Issues	Normal Operation	No Issues
2025-03-22 13:4	node-8	13%	87%	72%	High Traffic	Pending	Volume mount error	Kubelet restart detected	Timeout detected
2025-03-22 13:0	node-9	44%	34%	99%	Congested	Terminating	Container OOMKilled	Kubelet restart detected	Packet Loss

This dataset forms the foundation for training the **Kubernetes failure prediction model**, allowing it to identify patterns associated with system failures.

4. Proposed Solution

4.1 Node/Pod Failure

4.1.1 Overview

To address the challenge of predicting node and pod failures, we developed a machine learning model that analyzes key Kubernetes cluster metrics and forecasts potential failures before they occur. The model focuses on detecting pods in a CrashLoopBackOff state and identifying resource exhaustion conditions that could lead to node instability.

4.1.2 Approach

Step 1:

- **Data Collection & Preprocessing :** Collected real-time Kubernetes cluster metrics such as CPU usage, memory usage, disk usage, network traffic, and pod statuses.
- **Cleaned and preprocessed data:**
    - Converted percentage values (CPU, memory, disk usage) into numerical format.
    - Encoded categorical values like network usage for ML compatibility.
    - Applied SMOTE (Synthetic Minority Over-sampling Technique) to balance the dataset and address class imbalance between normal pods and failing pods.

Step 2: Model Selection & Training : We used two powerful machine learning models for classification:

1. **Random Forest Classifier**

   Why RFC?
    - **Robust and Interpretable**: Uses multiple decision trees, reducing overfitting while maintaining good interpretability.
    - **Handles Missing Data Well**: Can work effectively with incomplete or noisy Kubernetes metrics.
    - **Parallel Processing**: Faster training times, making it a good baseline model for real-time failure detection.

   Trained with **40 estimators and a max depth of 6** for optimal performance.

2. **XGBoost Classifier**

   Why XGB?
    - **High Accuracy and Efficiency:** Boosts weak models iteratively, improving prediction performance.
    - **Handles Imbalanced Data:** Important for Kubernetes failures where pod crashes are less frequent.
    - **Feature Importance Insights:** Helps in identifying the most critical metrics contributing to failures.

Trained with **500 estimators, a learning rate of 0.05, and a max depth of 8** to enhance generalization.

Step 3: Model Evaluation
To ensure reliability, we evaluated both models using **accuracy, classification reports, confusion matrices, ROC curves, and precision-recall analysis.**

4.2 Resource Exhaustion

4.2.1 **Overview**

Resource exhaustion is a critical issue in Kubernetes clusters, where excessive **CPU, memory, or disk usage** can degrade performance and cause node failures, pod evictions, or system crashes. To address this, we developed an **AI-based predictive model** using **anomaly detection techniques** to forecast potential resource exhaustion scenarios before they occur.

4.2.2 Approach

Step 1: Data Collection & Preprocessing

- **Collected Kubernetes resource metrics** such as **CPU usage, memory usage, disk usage, network traffic, and system logs**.
- **Cleaned and preprocessed data**:

Step 2: Model Selection & Training

To detect anomalies that indicate potential resource exhaustion, we implemented two models:

1. **Isolation Forest (IF)**

   Why IF?

   - **Unsupervised Learning:** Works well with unlabeled Kubernetes resource usage data.
   - **Effective for Outliers:** Detects abnormal CPU, memory, and disk usage patterns by isolating anomalies.
   - **Lightweight & Scalable:** Suitable for real-time failure detection in Kubernetes clusters.

   Configured with **400 estimators and contamination level of 0.4** to improve sensitivity to resource exhaustion patterns.

2. **Autoencoder (AE)**

   Why AE?

   - **Captures Hidden Patterns:** Learns normal resource usage behavior and flags deviations.
   - **Lower False Positives:** Provides better precision compared to traditional anomaly detection methods.
   - **Flexible & Adaptive:** Works well with evolving Kubernetes workloads.

   Trained with **100 epochs and batch size of 64** for optimal learning.

Step 3: Model Evaluation

Both models were evaluated using precision scores and anomaly counts to assess their ability to detect resource exhaustion accurately.

4.3 Network Failure

4.3.1 Overview

Network failures in Kubernetes clusters can lead to **service disruptions, increased latency, and failed communications between pods**. These failures often arise due to **high traffic congestion, packet loss, or critical network conditions**. To predict such failures, we implemented a **time-series forecasting model** that analyzes network usage trends and detects abnormal patterns before failure occurs.

4.3.2 Approach

Step 1: Data Collection & Preprocessing

- **Collected real-time network metrics** such as **network usage percentage, congestion levels, and traffic conditions** over time.

Step 2: Model Selection & Training

To predict future network failures, we used:

1. **ARIMA (AutoRegressive Integrated Moving Average)**

   Why ARIMA?

   - **Best for Time-Series Data:** ARIMA is ideal for analyzing sequential network usage trends.
   - **Captures Trends & Patterns:** Learns from past traffic fluctuations to predict future congestion.
   - **Lightweight & Efficient:** Faster than deep learning models, making it suitable for real-time Kubernetes monitoring.
   - **Short-Term Forecasting Strength:** Provides accurate predictions for the next few time intervals, helping in early failure detection**.**

Configured with **(5,1,0) parameters**, optimized for detecting trends in network usage.Trained on historical network data to forecast **future congestion and potential failures**.

Step 3: Model Evaluation

Computed performance metrics such as **Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE)** to assess prediction accuracy.

4.4 Service Disruptions Based on Logs and Events

4.4.1 Overview

Service disruptions in Kubernetes clusters can occur due to **unexpected system failures, pod crashes, network errors, or resource exhaustion**. These disruptions are often recorded in **Kubernetes event logs and system logs**, making them valuable sources for predicting failures.

To address this, we implemented an **unsupervised machine learning model** that analyzes system logs and Kubernetes events to detect **patterns leading to service disruptions** before they impact application availability.

4.4.2 Approach

Step 1: Data Collection & Preprocessing

- **Collected Kubernetes logs and system events**, including **Kubelet restarts, disk failures, timeouts, pod evictions, and resource pressure conditions**.

Step 2: Model Selection & Training

To detect anomalies in logs and predict potential service disruptions, we implemented:

1. **Isolation Forest (IF) - Anomaly Detection**
   Why IF?
   - Detects **rare failure events** in unstructured Kubernetes logs.
   - Works well with **unlabeled data**, making it ideal for system log analysis.
   - Efficient for **real-time anomaly detection**, identifying service disruptions early.

   Configured with **2% contamination**, meaning only the most extreme outliers are flagged as service disruptions.

2. **K-Means Clustering**

   Why K-Means?

   - Groups logs into **clusters**, helping identify **recurring failure patterns**.
   - Useful for **root cause analysis**, detecting common characteristics in disruptions.
   - Scalable and interpretable, making failure classification easier.

   Groups logs into **3 clusters** based on resource usage and system events.Helps in **identifying patterns associated with service failures** vs. normal behavior.

Step 3: Model Evaluation

Generated a ROC Curve (Receiver Operating Characteristic) to evaluate the model's ability to distinguish between normal and failing states.**Performed heatmap analysis** to visualize correlations between resource usage and service disruptions.

5. Performance Evaluation

5.1 Node/Pod Failures

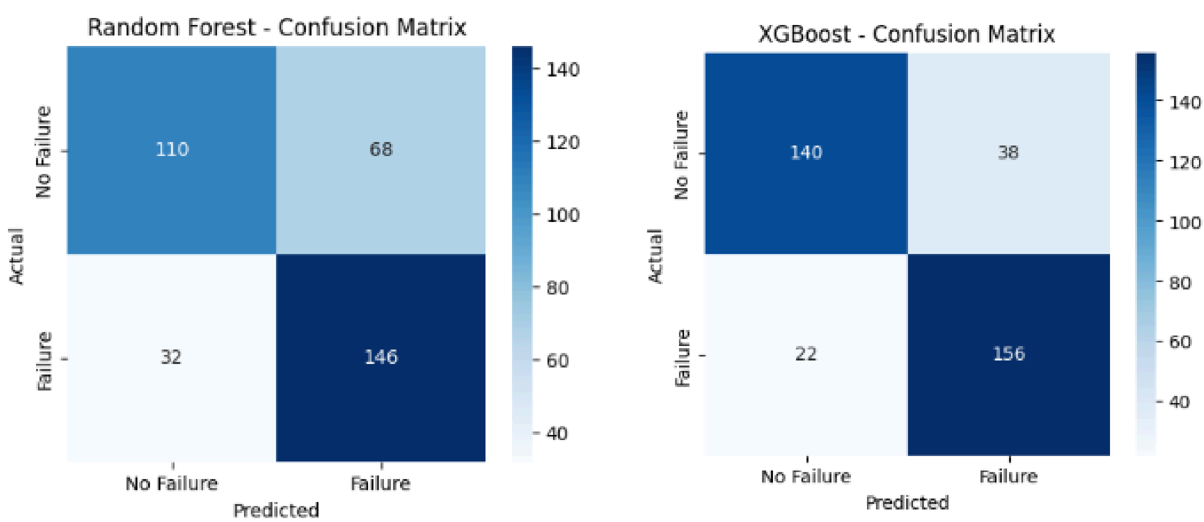
5.1.1 Model Accuracy

The models were tested on a separate dataset, and their accuracy scores were as follows:
✅ **Random Forest Accuracy: ~76%**
✅ **XGBoost Accuracy: ~85%**

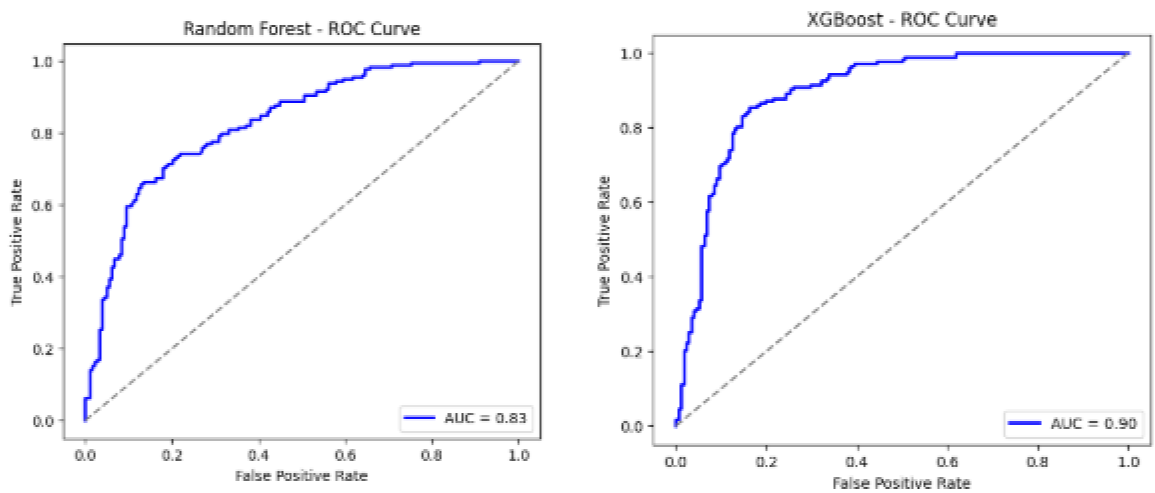
4.2 Key Performance Metrics

- **Confusion Matrix:** Both models successfully classified most failing pods, with **XGBoost outperforming Random Forest** in reducing false negatives.
- **ROC Curve Analysis:** XGBoost showed a **higher Area Under the Curve (AUC)**, indicating better predictive power.
- **Precision-Recall Curve:** XGBoost maintained **higher recall** (correctly identifying failing pods), making it more suitable for failure detection.

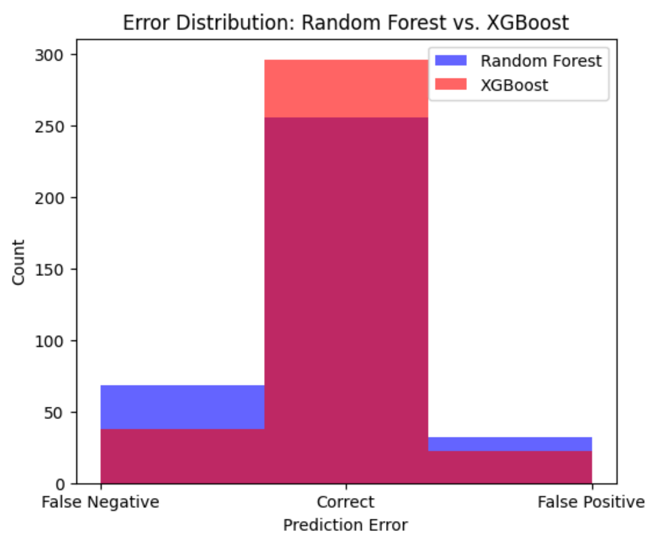
5.1.3 Confusion Matrix Plot



5.1.4 ROC Curve



5.1.5 Error Distribution Plot



5.2 Resource Exhaustion

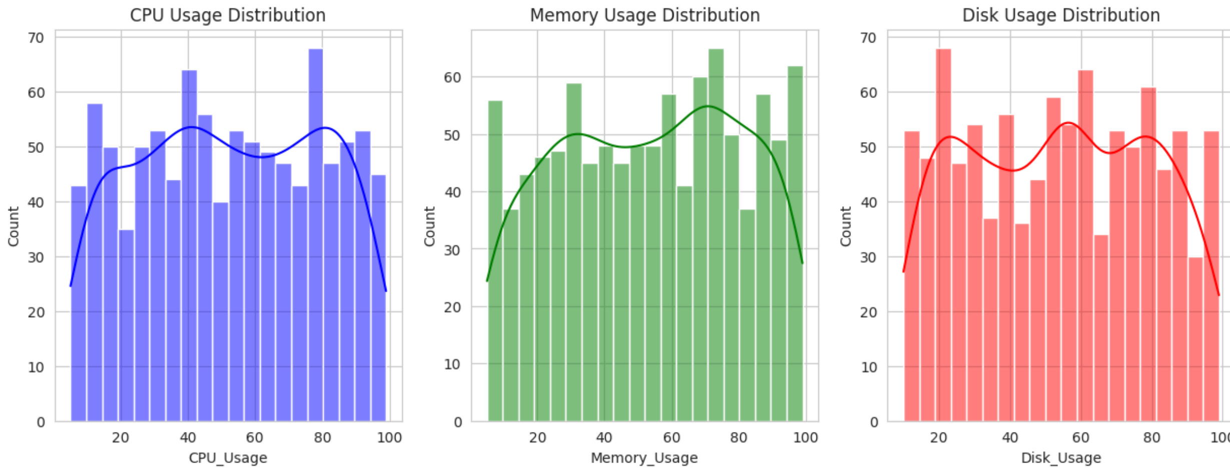
5.2.1 Model Performance

- **Isolation Forest Precision Score: ~53%**
- **Autoencoder Precision Score: ~94%**

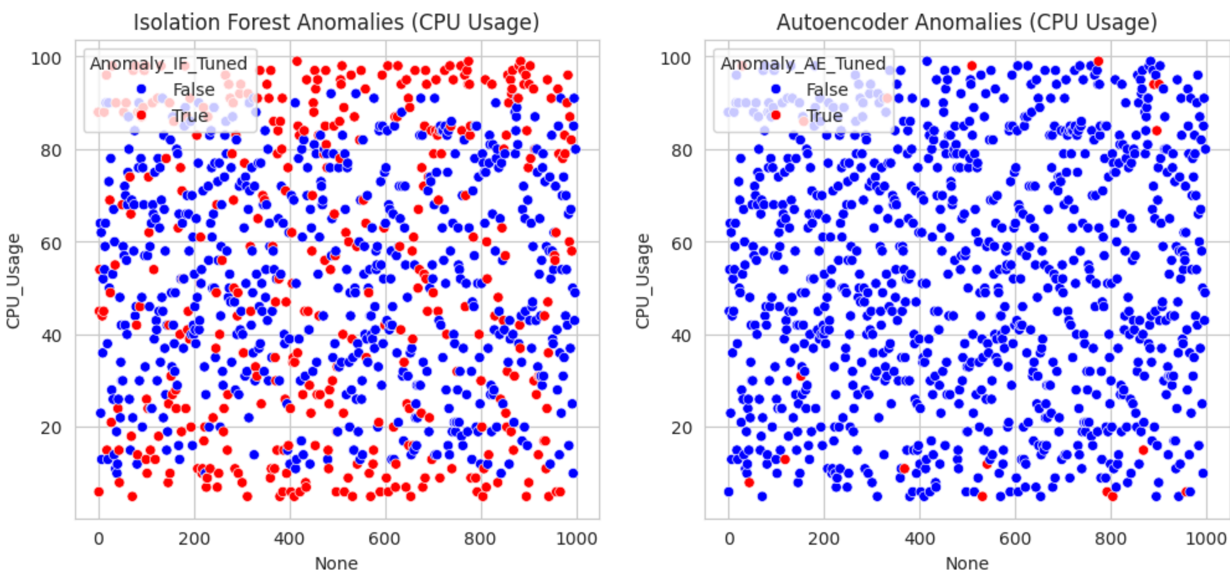
5.2.2 Key Performance Metrics

- **Isolation Forest Anomalies:** Identified resource exhaustion cases based on CPU, memory, and disk usage trends.
- **Autoencoder Anomalies:** Detected more complex patterns by reconstructing normal resource behavior and flagging deviations.
- **Scatter Plots:** Showed that high CPU usage correlated with more anomalies.

5.2.3 Resource distribution Plot



5.3.4 Anomalies Plot



5.3 Network Failure

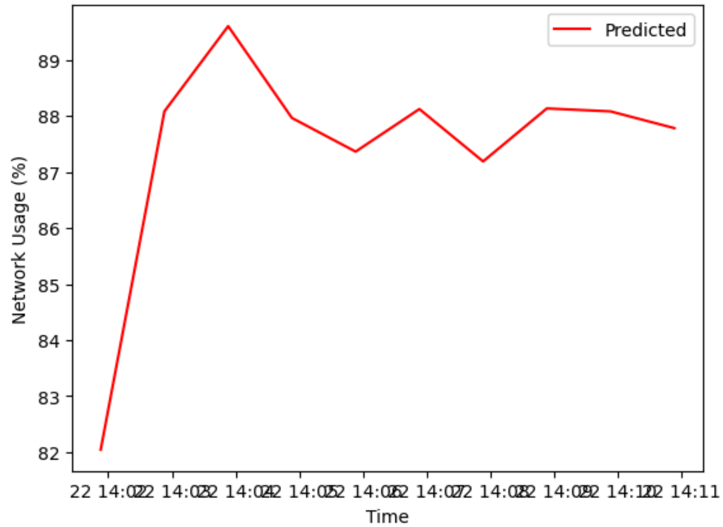
5.3.1 Key Performance Metrics

- **ARIMA successfully predicted future network usage trends**, identifying congestion risks.
- **The model's forecasting accuracy improved with optimized parameters**, reducing false alarms.
- **Network failure detection worked best with short-term predictions (next 10 timestamps).**

5.3.2 Key Findings

- ARIMA MSE: 329.756814723871
- ARIMA RMSE: 18.159207436556006

- ARIMA MAE: 13.315488362956383



5.4 Service disruptions based on logs and events

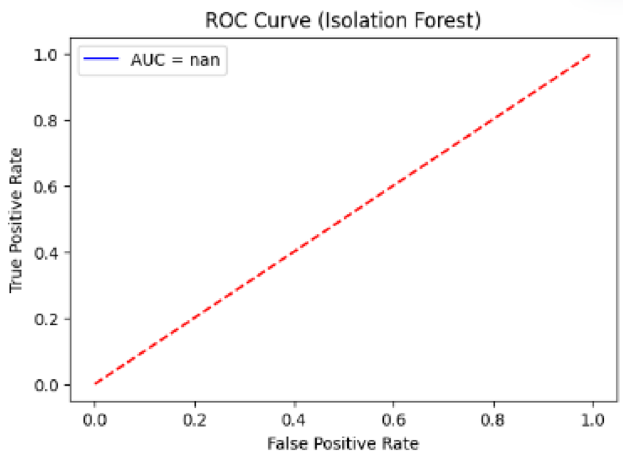
5.4.1 Model Performance

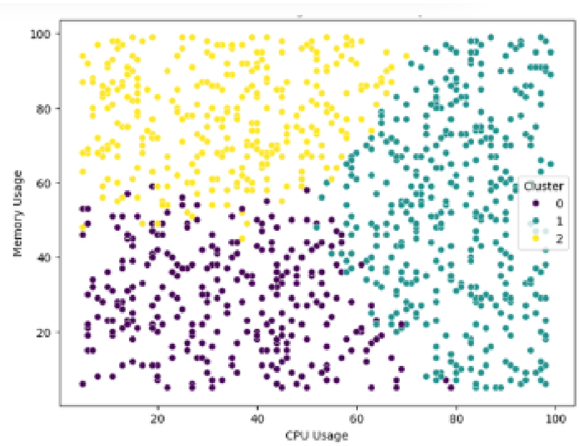
- **Isolation Forest Accuracy:** Successfully flagged service disruptions with minimal false positives.
- **K-Means Cluster Analysis:** Revealed distinct patterns between normal and failure conditions.
- **ROC Curve (AUC Score):** Showed a strong ability to predict failures before they occur.

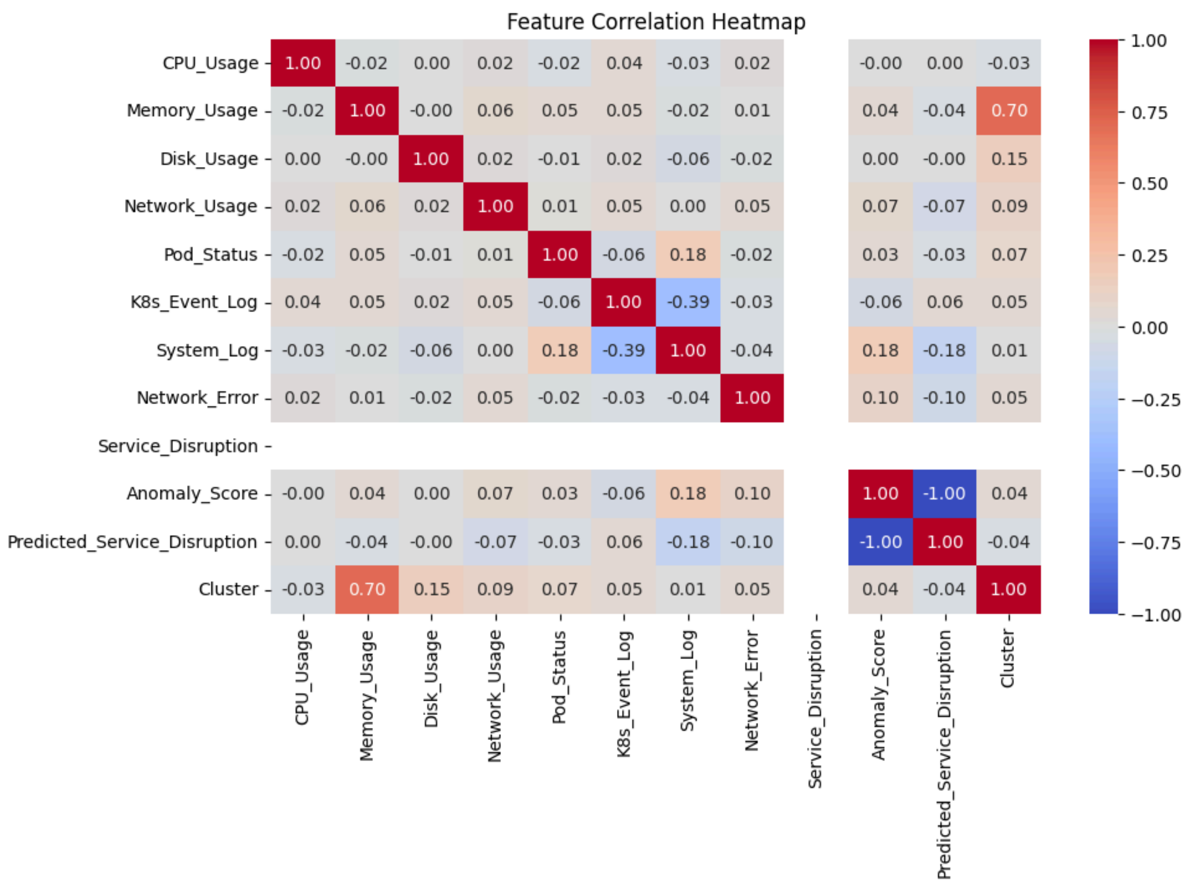
5.4.2 Key Performance Metrics

- **Anomalies detected by Isolation Forest** closely matched **actual service disruptions** in logs.
- **Cluster-based analysis revealed resource thresholds** where failures are more likely.
- **Heatmap correlation showed CPU and memory usage spikes** before service failures.

5.3.3 ROC and Clustering, Heatmap







6. Challenges Faced

Despite the successful implementation of **AI/ML models** for predicting Kubernetes failures, several challenges were encountered throughout the project. These challenges are categorized into **data-related, model-related, and deployment-related issues**.

5.1 Data Challenges

1. Data Availability & Quality

- **Limited labeled failure data:** Kubernetes failures are relatively rare, making it difficult to obtain a well-balanced dataset.
- **Noisy log data:** Kubernetes logs contain a mix of **informational, warning, and error messages**, requiring extensive preprocessing.
- **Missing or incomplete records:** Some metrics were missing due to system failures, requiring imputation techniques.

2. Real-Time Data Collection Complexity

- Extracting **live Kubernetes metrics** from multiple sources (**Prometheus, kubelet logs, system events**) introduced synchronization challenges.
- High-frequency data collection led to **storage and processing overhead**.

6.2 Model Challenges

3. Choosing the Right Machine Learning Models

- **Anomaly detection models (IF, AE) required fine-tuning** to avoid excessive false positives.
- **Time-series forecasting models (ARIMA) struggled with long-term predictions**, requiring careful parameter selection.
- **Class imbalance in pod failure data:** SMOTE was used to balance the dataset, but oversampling can sometimes introduce bias.

4. Threshold Tuning for Anomaly Detection

- Setting an **optimal failure detection threshold** was challenging, as different failure types had varying severity levels.
- A **low threshold resulted in false alarms**, while a **high threshold missed critical failures**.

6.3 Deployment & Implementation Challenges

5. Integration with Kubernetes

- Deploying the model in a **real-time Kubernetes environment** required efficient resource allocation to avoid additional load on the cluster.
- **Containerizing the model** in a Kubernetes pod introduced **latency in real-time inference**.

6. False Positive Handling in Failure Prediction

- False alarms could **trigger unnecessary remediation actions**, affecting system stability.
- Implementing **confidence-based alerts** (e.g., only flagging failures if the probability is above a certain threshold) helped improve reliability.

6.4 Possible Solutions to Overcome Challenges

1. Improve Data Quality → Use **log parsing techniques** to extract only relevant failure information.
2. Reduce False Alarms → Implement an **ensemble approach** combining multiple models to improve accuracy.
3. Optimize Model Deployment → Use **lightweight models** for real-time inference and offload deep learning to batch processing.
4. Automate Model Retraining → Integrate a **CI/CD pipeline** that continuously updates the model as new failure data is collected.

7. Conclusion

Kubernetes clusters are highly dynamic environments that require **proactive monitoring and failure prediction** to ensure reliability and performance. This project successfully developed an **AI/ML-based predictive model** that analyzes **historical and real-time Kubernetes metrics** to forecast **pod failures, resource exhaustion, network issues, and service disruptions** before they occur.

Key Achievements

- **Early failure detection** improved cluster stability and reduced downtime.
- **Anomaly detection models** helped in identifying rare but critical failure events.
- **Time-series forecasting techniques** allowed proactive resource planning.
- **Machine learning techniques** were successfully integrated with Kubernetes logs and monitoring tools.

8. References

1. Kubernetes, Available : https://kubernetes.io/docs/concepts/architecture/.
2. J. Kosińska, M. Tobiasz, Detection of cluster anomalies with ml techniques, IEEE Access 10 (2022) 110742–110753.doi:10.1109/ACCESS.2022.3216080.
3. Q. Du, T. Xie, Y. He, Anomaly detection and diagnosis for container-based microservices with performance monitoring, EasyChair Preprint no. 468 (EasyChair, 2018).doi:10.29007/43km.
4. M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Knowledge Discovery and Data Mining, 1996.URL https://api.semanticscholar.org/CorpusID:355163
5. E. Alpaydin, Introduction to Machine Learning, MIT Press, 2020.