**KUBERNETES FAILURE PREDICTION & REMEDIATION AGENT: DOCUMENTATION**

**PROJECT OVERVIEW**

Kubernetes clusters can encounter failures such as pod crashes, resource bottlenecks, and network issues. The challenge in Phase 1 is to build an AI/ML model capable of predicting these issues before they occur by analyzing historical and real-time cluster metrics. Once issues are predicted, the next step is to automate or recommend actions for remediation. The challenge in Phase 2 is to create an agent or system capable of responding to these predicted issues by suggesting or implementing actions to mitigate potential failures in the Kubernetes cluster.

**APPROACH**

**Problem Addressed**

Kubernetes clusters are susceptible to a variety of unpredictable failures, such as pod crashes, resource bottlenecks, and network issues. These failures can lead to downtime, degraded performance, and increased operational overhead. Currently, detecting and responding to failures is often a manual process, which is time-consuming and error-prone.

The solution is to build an **AI-powered system** that:

* **Predicts failures** proactively by analyzing historical and real-time metrics.
* Provides **explanations** for the predictions through **Explainable AI (XAI)**.
* **Recommends or triggers automated remediation actions** to resolve failures.

**Key Metrics Collected**

The system collects a wide array of **real-time metrics** from Kubernetes pods to train the failure prediction model. These metrics provide insights into the health and behavior of pods, and help detect potential issues before they escalate.

**Metrics Collected:**

1. **CPU Usage** (cpu\_usage\_cores): Tracks the amount of CPU resources used by a pod.
2. **Memory Usage** (memory\_usage\_bytes): Monitors the memory consumption of a pod.
3. **Disk I/O** (disk\_io\_read/write\_bytes): Measures the read/write activity on the pod’s disk.
4. **Network I/O** (network\_rx/tx\_bytes): Tracks network traffic, both incoming and outgoing.
5. **Latency** (latency\_ms): Measures the response time or delay in pod operations.
6. **Pod Health Metrics**:
   * restart\_count: The number of times a pod has been restarted.
   * container\_ready: Indicates whether the container within a pod is ready for service.
   * pod\_scheduled: The state of pod scheduling within the cluster.

**Error Logs & Failure Scenarios:**

In addition to the metrics, simulated error logs reflecting real-world failures were collected:

* **OOM kill** (Out of Memory Kill)
* **Container crash**
* **CPU throttle**
* **DNS failure**
* **Disk full**
* **Network loss**
* **Kubelet down**

These failure events are crucial for simulating realistic Kubernetes scenarios and training the failure prediction model effectively.

**Chaos Engineering Approach:**

To simulate real-world conditions, we incorporated **chaos engineering** principles into our system. We injected faults such as **CPU throttling**, **memory exhaustion**, **kubelet failures**, and **DNS issues** into the system. This chaos was introduced to assess how the system behaves under stress and enable the model to learn from fault-induced episodes, ultimately improving the model’s accuracy and resilience.

**MODEL DEVELOPMENT & PERFORMANCE**

**Preprocessing Pipeline**

The **data preprocessing pipeline** is designed to ensure that the collected metrics are in the correct format for machine learning modeling. Key preprocessing steps include:

* **Timestamp Parsing & Sorting**: Ensures that data is properly ordered to maintain temporal integrity.
* **Label Encoding**: Categorical features like namespace and failure\_type are encoded for machine learning models.
* **StandardScaler**: Numerical metrics are normalized to standardize the data.
* **Train-Test Split**: The dataset is split into training and testing sets (80/20 split) with **time-based shuffling** to avoid data leakage.

**Model Selection**

For the failure prediction model, we selected the **XGBoostClassifier**, which is known for its robustness in handling tabular data and imbalanced datasets. This model is particularly suited for real-time predictions in environments where rare events (such as pod failures) need to be detected with high recall.

**Model Tuning:**

* **Threshold Tuning**: To improve recall, especially for rare failure types, threshold tuning was applied. This helped maximize the model’s ability to catch **true failures** early, even at the cost of reduced accuracy.
* **Model Output**: The model predicts the probability of failure (fail/no-fail) along with an associated risk score for each pod, categorized into risk levels: **High**, **Medium**, **Low**, and **None**.

**Model Performance:**

While **accuracy** in traditional machine learning models is often a key performance metric, for this use case, **recall** is the most important metric due to the nature of the task. In Kubernetes environments, failures are rare, and predicting a "no failure" scenario would yield high accuracy but provide no real value.

**Key Model Metrics**:

* **Accuracy**: ~60% (due to imbalanced dataset)
* **Recall**: Focused on maximizing recall to catch early failures.
* **Precision**: Balanced with recall through threshold tuning.
* **Risk Levels**: The predicted failure likelihood is associated with risk levels (High, Medium, Low, None).

**Explainable AI (XAI):**

To ensure transparency and trust in the model, **Explainable AI (XAI)** is integrated. Each failure prediction comes with an explanation that details:

* **Why** the pod is likely to fail (e.g., "High CPU usage and OOM kill detected").
* **Feature contributions** to the prediction (e.g., SHAP values or rule-based reasoning).

These explanations help operations teams understand the causes behind predictions and ensure the system’s decisions can be interpreted.

**REMEDIATION ACTIONS**

**Action Triggering Logic**

Once a failure prediction is made, the system automatically suggests or triggers **remediation actions** to resolve the predicted issue before it leads to downtime or performance degradation.

**Failure Types and Corresponding Remediation Actions:**

1. **Pod Stuck** → **Restart the pod**: If a pod is not progressing or stuck in a state, it is restarted to restore functionality.
2. **CPU Throttling** → **Scale CPU resources**: If a pod is being throttled due to insufficient CPU resources, scaling up the pod’s CPU allocation resolves the issue.
3. **Memory Leak** → **Increase memory allocation**: If a pod is experiencing memory leakage, increasing memory resources prevents the pod from being killed due to memory exhaustion.
4. **Disk Full** → **Clean storage**: If disk usage exceeds capacity, the system recommends cleaning or increasing storage resources.
5. **Default Action**: In cases where no specific remediation action is defined, the system sends an **alert to the admin** with failure details and suggested next steps.

**Remediation in Action**

* **Simulated Kubernetes Actions**: The system can generate **kubectl commands** or **YAML configurations** to automate remediation in real Kubernetes clusters.
* **Webhook Integration**: In real-time systems, the remediation actions are sent as **webhooks** (e.g., to Slack or Discord) to notify administrators immediately.

**Remediation Flow:**

1. Failure prediction is made with associated risk score.
2. Based on the failure type, an appropriate remediation action is suggested.
3. The remediation action is displayed in the UI and can be executed manually or automatically.

**DEPLOYMENT AND PACKAGING**

To ensure the system is easily deployable within any Kubernetes environment:

* The **Streamlit app** is packaged into a **Docker container** for easy deployment.
* Kubernetes manifests (deployment.yaml, service.yaml) and a **Helm chart** are provided for easy deployment in **Kubernetes clusters**.
* The system was tested locally on **Minikube** to ensure compatibility.

**CONCLUSION**

This project delivers an intelligent, AI-driven system for predicting pod-level failures in Kubernetes environments, combined with an automatic remediation system to prevent downtime and improve operational efficiency. With a focus on **failure prediction**, **explainability**, and **automated remediation**, this solution empowers DevOps teams to proactively manage Kubernetes clusters and ensure high availability.

**Deliverables Summary**

1. **AI-powered failure prediction model** (XGBoostClassifier)
2. **Telemetric data collection and fault injection system**
3. **Streamlit UI** for interacting with predictions and remediation
4. **Kubernetes deployment** with Docker, Helm, and YAML configurations
5. **Remediation actions** and automatic execution
6. **Documentation and demo presentation**