BANGALORE INSTITUTE OF TECHNOLOGY
K.R. Road, V.V. Pura, Bengaluru-560 004

Hackathon Phase 2
Remediation for Predicted Kubernetes Issues

Submitted by

Team Name
“BIT MAVERICKS”

Team Members

Abhiraam S
Ranjan P
Darshan Sunil Kuranagi
Abhishek R
Dornadula Sai Gagan

GUIDEWIRE
DEVTrails
University Hackathon

TABLE OF CONTENTS

1. INTRODUCTION

1.1 BACKGROUND

Kubernetes has emerged as the leading platform for orchestrating containerized applications, enabling scalable and resilient cloud-native architectures (Kubernetes Overview). However, managing Kubernetes clusters at scale presents significant challenges, including detecting and mitigating security threats and operational anomalies in real-time. Manual monitoring often falls short due to the dynamic and distributed nature of containerized environments, where issues like Distributed Denial of Service (DDoS) attacks, resource exhaustion, or policy misconfigurations can lead to downtime or breaches.

The "Kubernetes Remediation Phase 2" project tackles these challenges head-on with an AI-driven system that autonomously detects anomalies and applies remediation actions to maintain cluster health and security. Designed for the Guidewire Solutions hackathon, this project integrates advanced technologies like Apache Kafka, Prometheus, Istio, and either LangChain or n8n to deliver a state-of-the-art solution. Imagine a system that acts like a vigilant guardian for your Kubernetes cluster: it watches over metrics collected from Prometheus, visualizes them with Grafana, and tracks network flows with IPTables-Netflow. An intelligent AI Agent then steps in, analyzing the incoming data—whether it’s CPU usage, network traffic, or logs—and redirects it to the perfect anomaly detection model for the job. For instance, a spike in network requests might signal a DDoS attack, while a memory surge could indicate resource exhaustion.

Here’s where it gets exciting; the AI Agent, whether powered by LangChain’s natural language processing capabilities or n8n’s workflow automation, sorts the data by type and routes it to specialized models tailored to predict specific issues. Once an anomaly is detected—say, a DDoS attack—the prediction is streamed through Kafka to a dedicated remediation agent. This agent springs into action, perhaps scaling up application instances or throttling traffic, resolving the issue before it escalates. Each detection model has its own remediation counterpart, ensuring precise, automated fixes in real-time.

This self-healing approach is a game-changer for the hackathon, blending cutting-edge AI with robust DevOps tools to create a scalable, efficient, and proactive system. This report dives into the project’s objectives, architecture, implementation, and evaluation, showcasing its innovation and potential to revolutionize Kubernetes cluster management.

1.2 OBJECTIVES

This 45-day hackathon challenge invites participants to develop a model that can predict potential issues in Kubernetes clusters (Phase 1) and recommend or automatically implement solutions to address those issues (Phase 2). The aim is to optimize the efficiency of Guidewire solutions on Cloud Infrastructure, using only publicly available resources and open-source solutions.

1.3 PHASE 2: REMEDIATION FOR PREDICTED ISSUES

1.3.1. Phase 2 Problem Statement Explanation:

Kubernetes clusters are complex systems that manage numerous containers across distributed nodes. This complexity introduces vulnerabilities that can disrupt operations, including:

- **Security Threats:** Issues like DDoS attacks, port scans, ICMP floods, and lateral movement attempts can exploit weaknesses in the network or applications, threatening the cluster's integrity.
- **Operational Issues:** Resource exhaustion (e.g., overuse of CPU or memory) and policy misconfigurations can degrade performance, disrupt services, or even cause outages.
- **Monitoring Challenges:** Traditional monitoring tools often fail to provide real-time insights across short-lived (ephemeral) containers, making it hard to respond quickly to emerging issues.

The challenge in Phase 2 is to develop a system or agent that takes the predictions from Phase 1 and either recommends or automatically implements actions to address these issues before they escalate. The goal is to create a scalable, automated solution that reduces the mean time to resolution (MTTR) and enhances the overall reliability of the Kubernetes cluster.

1.3.2. Key Objectives for Phase 2 :

The problem statement in the PDF outlines specific objectives for Phase 2, which are designed to tackle the complexities of Kubernetes management:

i) Remediation Actions : The system must respond to predicted issues with appropriate actions. Examples include:

- **Scaling pods:** Increasing the number of pods when resource exhaustion (e.g., CPU or memory shortages) is forecasted.

- **Restarting or relocating pods:** Addressing predicted pod failures by restarting them or moving them to healthier nodes.
- **Optimizing resource allocation:** Adjusting CPU or memory limits to prevent bottlenecks and ensure smooth operation.

ii) Automation :

The remediation system must integrate with the AI/ML agent from Phase 1 to trigger automatic responses. This means that when the Phase 1 model predicts an issue—like a resource bottleneck—the Phase 2 system should act without requiring human intervention, streamlining the process.

iii) Evaluation of Effectiveness :

It's not enough to simply apply remediation actions; the system must measure how well these actions work. This involves assessing whether the interventions successfully mitigate or prevent the predicted issues, ensuring the solution is reliable and efficient.

iv) Optional Kubernetes Packaging :

Participants are encouraged to package the remediation system in Kubernetes itself, demonstrating a fully integrated, cloud-native solution that can execute within the cluster environment.

Why Does This Matter?

Kubernetes clusters power many large-scale applications, and their complexity makes manual management impractical. Automated remediation addresses this by:

- **Reducing Downtime:** Quick responses to predicted issues minimize service disruptions.
- **Improving Reliability:** A proactive system keeps the cluster stable and secure.
- **Enhancing Scalability:** Automation allows the solution to handle growing clusters without added human effort.

By solving this problem, participants contribute to a self-healing Kubernetes environment, aligning with the hackathon's focus on innovative, cloud-native technologies.

1.3.3. Deliverables for Phase 2 :

To meet the Phase 2 challenge, participants must produce:

- **Remediation System:** A working system or agent that recommends or automates remediation based on Phase 1 predictions.
- **Codebase:** Functional code, uploaded to GitHub, that implements the remediation logic and connects to the prediction model.
- **Documentation:** Detailed explanation of how remediation actions are selected or executed.
- **Presentation:** A final presentation (with a recorded demo) showcasing the integrated prediction and remediation solution.
- **Optional Deployment:** A deployed application on a cloud platform, if feasible, for live testing.

2. SCOPE

Our solution combines AI-driven anomaly detection with automated remediation to proactively resolve issues in Kubernetes clusters. It integrates cutting-edge tools like LangChain, n8n, Apache Kafka, Prometheus, Grafana, IPTables-Netflow, and Istio to deliver an end-to-end, intelligent remediation system.This report provides a detailed account of our journey throughout the hackathon and is organized into the following key sections:

2.1. PROBLEM STATEMENT :

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.

2.2. METHODOLOGY:

i) Data Collection & Monitoring

- Prometheus + Grafana: Monitor real-time metrics (CPU, memory, pod status).
- IPTables-Netflow: Capture network traffic patterns.
- Centralized Logging: Aggregate structured/unstructured logs for AI analysis.

ii) AI-Driven Anomaly Detection

- LangChain: NLP-based log interpretation to identify failure patterns.
- n8n: Workflow engine for routing structured metrics to the right ML models.

iii)Model Types

- LSTM (Pod failures)
- ANN (Network anomalies)
- CNN (Resource exhaustion)
- Outlier/NLP models for logs

iv) Real-Time Streaming via Kafka

- Kafka pipelines connect anomaly predictions to remediation agents.
- Topics track anomaly alerts and executed actions for traceability.

v) Automated Remediation Engine

Dedicated agents respond to each issue type:

- Scaling pods for load or DDoS
- Restarting unstable workloads
- Blocking IPs or isolating nodes
- Kubernetes APIs and Istio policies enforce actions securely.

vi) Service Mesh with Istio

- Enables secure mTLS traffic, circuit-breaking, and traffic redirection during incidents.
- Adds observability for enhanced remediation decisions.

vii) Why It's Unique

- Modular and extensible architecture
- Fully Kubernetes-native deployment
- Combines NLP + metrics + automation for intelligent remediation
- Real-time, low-latency response loop using Kafka

viii) Impact

- Reduced MTTR (Mean Time to Recovery)
- Proactive issue resolution before escalation
- Autonomous infrastructure healing.

3. LITERATURE REVIEW AND RELATED WORK

3.1. EXISTING SYSTEMS & LIMITATIONS

Kubernetes clusters today are monitored using tools like Prometheus, Grafana, ELK Stack, and Istio. Prometheus scrapes metrics, while Grafana visualizes them through dashboards. The ELK stack aggregates logs, and Istio provides service mesh security and traffic control. However, these systems are largely reactive. Kubernetes health checks can restart failed pods, but they cannot predict failures based on early signs. Log aggregators like ELK do not interpret logs using NLP, making it difficult to extract actionable insights. Furthermore, these platforms often lack automation and rely on manual remediation, leading to slower recovery times. While commercial AIOps tools like Dynatrace offer AI-driven insights, they are expensive, closed-source, and not seamlessly integrated with Kubernetes-native workflows. Overall, these limitations highlight a gap in intelligent, proactive, and autonomous failure management systems.

3.2. PROPOSED APPROACH & CONTRIBUTIONS

Our solution addresses these gaps with a hybrid, AI-driven remediation framework. It combines LangChain for NLP-based log interpretation, n8n for workflow automation, and Kafka for real-time streaming. Each anomaly type—pod failure, network disruption, or resource exhaustion—is detected using specialized models (LSTM, ANN, CNN respectively). Once an issue is predicted, remediation agents automatically execute actions such as restarting pods, scaling deployments, or applying Istio-based traffic controls. Kafka ensures low-latency communication, while Istio enforces secure and intelligent routing. This approach enables predictive remediation, reduces manual effort, and minimizes downtime. Its modular, Kubernetes-native design makes it scalable, adaptable, and ideal for modern cloud-native infrastructures.

4. METHODOLOGY

The Kubernetes Remediation Phase 2 project presents a cutting-edge solution that synergizes artificial intelligence, real-time data processing, and Kubernetes-native automation to proactively manage and resolve anomalies in Kubernetes clusters. This approach minimizes human intervention, enhances scalability, and ensures robust cluster management, making it a standout entry for the hackathon. The solution integrates LangChain for intelligent log analysis, n8n for workflow automation, Apache Kafka for real-time streaming, Prometheus, Grafana, and IPTables-Netflow for metrics collection, Istio for traffic management, and specialized remediation agents to address detected issues. Below is a comprehensive breakdown of the approach.

4.1. Data Collection and Monitoring

The foundation of the solution lies in its ability to collect and monitor a diverse set of data sources within the Kubernetes environment:

- **Prometheus for Metrics Collection:** Prometheus scrapes real-time metrics from the Kubernetes cluster, such as CPU utilization, memory usage, pod health, and network I/O. These metrics form the backbone of anomaly detection by providing quantitative insights into cluster performance.
- **Grafana for Visualization:** Grafana complements Prometheus by offering real-time dashboards that visualize metrics, enabling operators to monitor trends and spot potential issues at a glance.
- **IPTables-Netflow for Network Insights:** IPTables-Netflow captures detailed network flow data, such as packet rates, source/destination IPs, and port usage. This is critical for identifying network-related anomalies like DDoS attacks or unauthorized access attempts.
- **Log Aggregation:** Logs from applications, Kubernetes components, and services are aggregated into a centralized system for further analysis.

This multi-faceted data collection strategy ensures that the AI Agent has a rich dataset to analyze, covering both structured metrics and unstructured logs.

4.2. AI-Driven Anomaly Detection

The AI Agent is the intelligent core of the system, leveraging LangChain and n8n to process data, classify its type, and route it to specialized anomaly detection models. Each model is paired with a corresponding remediation agent to resolve identified issues.

4.3. AI Agent Design

LangChain for Log Interpretation: LangChain, a framework for language model applications, empowers the AI Agent to process unstructured log data using natural language processing (NLP). The agent interprets log entries—such as error messages, warnings, or security alerts—and identifies patterns or keywords indicative of anomalies. For example:

- A log entry like "Failed to authenticate user from IP 192.168.1.100" might trigger a security anomaly model.
- Repeated "Out of memory" errors could signal resource exhaustion.
- n8n for Workflow Automation: n8n, an open-source workflow automation tool, handles structured metrics from Prometheus, Grafana, and IPTables-Netflow. It creates dynamic workflows to:

Classify Data: n8n examines the source and nature of the metrics (e.g., CPU spikes, network traffic surges) and determines the appropriate anomaly detection model.

Route Data: Based on classification, n8n directs the data to the relevant model. For instance, high network packet rates are sent to a DDoS detection model, while pod crash metrics go to a stability model.

4.4. Specialized Anomaly Detection Models

The system employs multiple machine learning models, each tailored to a specific anomaly type:

- **Time-Series Forecasting:** Analyzes metrics like CPU and memory usage to predict resource exhaustion.
- **Outlier Detection:** Identifies unusual patterns in network traffic or pod behavior, such as a sudden spike in requests from a single IP.
- **NLP-Based Log Analysis:** Detects error patterns or security events in logs, such as repeated authentication failures.

This hybrid approach—combining LangChain’s language understanding with n8n’s automation—ensures comprehensive coverage across diverse data types, making the anomaly detection process both intelligent and efficient.

4.5. Real-Time Data Streaming with Apache Kafka

Once an anomaly is detected, the prediction is streamed in real-time via Apache Kafka:

- **Anomaly Prediction Topic:** Each detection model publishes its predictions (e.g., "DDoS detected," "Resource limit exceeded") to a dedicated Kafka topic. This ensures low-latency communication between detection and remediation components.
- **Remediation Log Topic:** A separate Kafka topic logs all remediation actions, providing an audit trail for post-incident analysis and compliance.

Kafka's distributed architecture guarantees scalability and reliability, enabling the system to handle high volumes of predictions without performance degradation.

4.6. Automated Remediation

For every anomaly type, a dedicated Remediation Agent subscribes to the Kafka anomaly prediction topic and executes context-aware actions:

-Remediation Agent Workflow:

- When a prediction is received, the agent matches the anomaly type to its predefined action set.
- Actions are executed using Kubernetes APIs and Istio configurations.

- Examples of Remediation Actions:

- **DDoS Attack:** The agent scales up the affected deployment to distribute load and uses Istio to apply rate limiting or block malicious IPs.
- **Resource Exhaustion:** It adjusts resource limits, scales pods horizontally, or restarts failing containers.
- **Security Threat:** The agent isolates compromised pods, blocks suspicious IPs via network policies, or enforces stricter authentication rules.

This one-to-one mapping between detection models and remediation agents ensures precise, tailored responses, avoiding generic fixes that might overlook root causes.

4.7. Service Mesh with Istio

Istio enhances the solution with advanced traffic management and security:

- **Mutual TLS (mTLS):** Encrypted communication between services, reducing the risk of interception or tampering.
- **Traffic Control:** Features like load balancing, circuit breaking, and dynamic routing support remediation actions, such as redirecting traffic away from failing pods.

By integrating Istio, the solution ensures that remediation actions are executed with precision and security, maintaining cluster integrity even under attack.

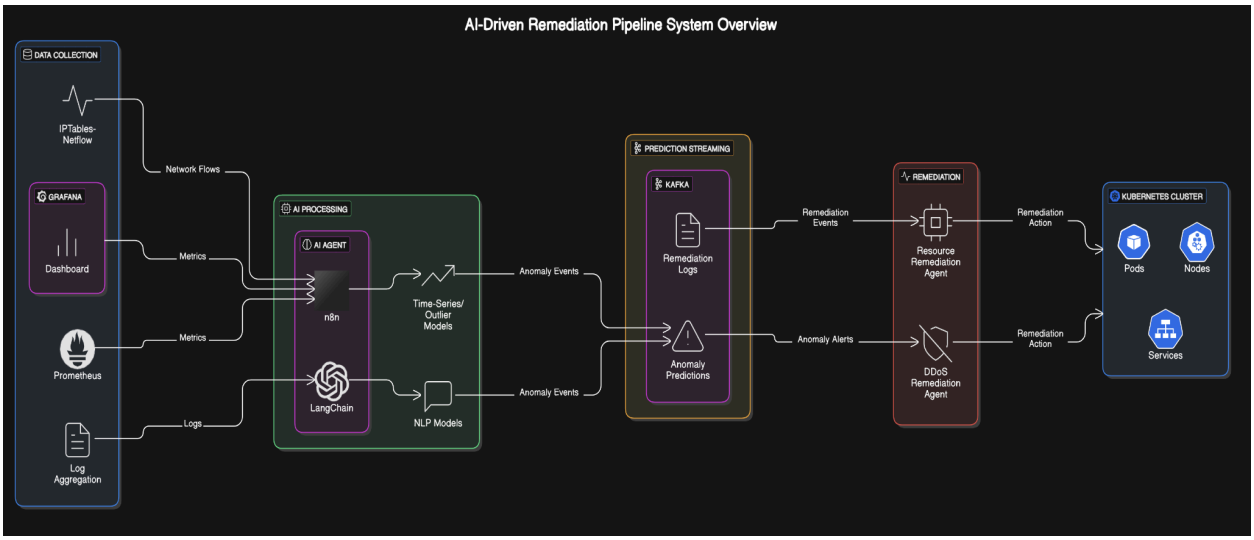
4.8. Why This Solution is State-of-the-Art

This approach pushes the boundaries of Kubernetes cluster management with several innovative features:

- **Intelligent Data Processing:** The combination of LangChain's NLP and n8n's workflow automation creates an AI Agent that seamlessly handles both logs and metrics, routing them to the right models with minimal latency.
- **Real-Time Responsiveness:** Kafka's streaming capabilities ensure that anomalies are addressed as soon as they're detected, reducing downtime and risk.
- **Modular and Scalable Design:** Separate detection models and remediation agents allow for easy expansion—new anomaly types can be added without overhauling the system.
- **Kubernetes-Native Integration:** Built to run within Kubernetes, the solution leverages its orchestration power and integrates with cloud-native tools like Prometheus and Istio, ensuring portability across environments.
- **Automation Excellence:** By automating detection and remediation end-to-end, the system slashes mean time to resolution (MTTR) and frees DevOps teams for higher-value tasks.

5. SYSTEM ARCHITECTURE

5.1. ARCHITECTURE DIAGRAM



5.2. MODULE OVERVIEW

Our solution consists of modular components designed for proactive Kubernetes remediation. Metrics are collected via Prometheus, logs are aggregated centrally, and network data is captured using IPTables-Netflow. This data is preprocessed and sent to specialized AI models: LSTM for pod failure prediction, ANN for detecting network issues, and CNN for identifying resource exhaustion. LangChain is used to interpret unstructured logs through NLP, enabling intelligent log analysis. The n8n workflow engine routes data dynamically to the appropriate model or remediation logic. Apache Kafka handles real-time streaming between modules. Once an issue is detected, remediation agents execute automated actions through Kubernetes APIs and Istio, ensuring secure and immediate response.

5.3. UNIQUE ASPECT OF OUR SYSTEM

Our project stands out by combining AI, automation, and real-time orchestration in a Kubernetes-native solution. Unlike typical monitoring tools, we use different machine learning models for specific issue types, increasing accuracy. The use of LangChain for NLP-based log understanding is a novel addition, enabling insights from raw logs. The integration of n8n allows for dynamic, automated routing and decision-making. Real-time processing with Kafka and autonomous remediation using Kubernetes and Istio complete a highly responsive, scalable, and intelligent framework built entirely on open-source tools.

6. IMPLEMENTATION

Implementation Details: This document explores the implementation of the Kubernetes Remediation Phase II project, an AI-driven system for detecting and remediating anomalies in Kubernetes clusters. The focus is on the remediation agent's logic (written in Python), its deployment configuration (via YAML), and the environment setup steps. Below are the key code snippets with detailed explanations.

1. Remediation Agent Logic (remediation_agent.py)

The remediation_agent.py file is the heart of the remediation system, handling anomaly detection responses and remediation actions.

```
Kafka Consumer Initialization

self.consumer = KafkaConsumer(

    'predictions',

    bootstrap_servers=[self.kafka_server],

    value_deserializer=lambda x: json.loads(x.decode('utf-8')),

    auto_offset_reset='earliest',

    group_id='remediation-agent'

)
```

Sets up a Kafka consumer to listen for anomaly predictions published to the predictions topic.

Explanation:

- predictions: The Kafka topic where anomaly detection messages are sent.
- bootstrap_servers: Connects to the Kafka server (e.g., kafka-0.kafka-headless.default.svc.cluster.local:9092).
- value_deserializer: Converts incoming JSON messages into Python dictionaries.
- auto_offset_reset='earliest': Reads all messages from the topic’s beginning.
- group_id: Assigns the consumer to the remediation-agent group for coordination.

```
Kafka Producer Initialization

self.producer = KafkaProducer(

    bootstrap_servers=[self.kafka_server],

    value_serializer=lambda v: json.dumps(v).encode('utf-8')

)
```

Purpose: Configures a Kafka producer to send remediation logs to the remediation_logs topic.

Explanation:

- value_serializer: Serializes Python dictionaries into JSON-encoded messages.
- This producer is used to log the success or failure of remediation actions.

1.2. DDoS Remediation Logic:

```
def handle_ddos(self):

    try:

        self.apps_v1.patch_namespaced_deployment_scale(

            name="test-app",

            namespace="default",

            body={"spec": {"replicas": 3}}

        )

        self.log_remediation("DDoS", "success")

    except Exception as e:

        logger.error(f"DDoS remediation failed: {str(e)}")

        self.log_remediation("DDoS", "failed")
```

Purpose: Scales the test-app deployment to 3 replicas when a DDoS anomaly is detected.

Explanation:

- patch_namespaced_deployment_scale: Uses the Kubernetes Python client to update the deployment's replica count.
- name="test-app": Targets the test application deployment.
- namespace="default": Operates in the default namespace.
- body={"spec": {"replicas": 3}}: Specifies the desired replica count.
- log_remediation: Logs the outcome (success or failure) to Kafka.

1.3. Main Loop

```
def run_kafka(self):

    while True:

        try:

            if not self.consumer:

                self.initialize_kafka()

            for message in self.consumer:

                anomaly = message.value

                if anomaly.get("anomaly_type") == "DDoS":

                    self.handle_ddos()

        except KafkaError as e:

            logger.error(f"Kafka consumer error: {str(e)}")

            self.consumer = None

            time.sleep(5)
```

Purpose: Continuously listens for Kafka messages and triggers remediation based on anomaly type.

Explanation:

- Infinite loop ensures the agent stays active.
- Reinitializes Kafka if the consumer fails.
- Checks each message for anomaly_type == "DDoS" and calls handle_ddos() accordingly.
- Includes error handling with a 5-second delay before retrying.

2. Deployment Configuration (agent-deployment.yaml)

The YAML configuration deploys the remediation agent and sets up its permissions.

```
Agent Deployment

apiVersion: apps/v1

kind: Deployment

metadata:

  name: remediation-agent

  namespace: default

spec:

  replicas: 1

  selector:

    matchLabels:

      app: remediation-agent

  template:

    metadata:

      labels:

        app: remediation-agent

    spec:

      serviceAccountName: remediation-agent

      containers:

      - name: remediation-agent
```

```
        image: remediation-agent:v1

        imagePullPolicy: Never
```

Purpose: Deploys the remediation agent as a single pod in the Kubernetes cluster.

Explanation:

- replicas: 1: Runs one instance of the agent.
- serviceAccountName: remediation-agent: Links to a custom service account for RBAC.
- image: remediation-agent:v1: Uses the locally built Docker image.
- imagePullPolicy: Never: Prevents Kubernetes from pulling the image remotely, assuming it's built locally.

2.1.RBAC ClusterRole

```
apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

  name: remediation-agent-role

rules:

- apiGroups: ["apps"]

  resources: ["deployments", "deployments/scale"]

  verbs: ["get", "list", "watch", "update", "patch"]
```

```
Purpose: Grants the agent permissions to manage deployments.

Explanation:

apiGroups: ["apps"]: Targets deployment-related resources.

resources: Allows interaction with deployments and their scale
subresource.
```

```
verbs: Permits reading (get, list, watch) and modifying (update,
patch) actions.
```

2.2. ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

  name: remediation-agent-binding

subjects:

- kind: ServiceAccount

  name: remediation-agent

  namespace: default

roleRef:

  kind: ClusterRole

  name: remediation-agent-role

  apiGroup: rbac.authorization.k8s.io
```

Purpose: Binds the remediation-agent-role to the remediation-agent service account.

Explanation:

Links the permissions defined in the ClusterRole to the agent's service account, enabling it to perform remediation actions.

3. Environment Setup Commands:

The setup guide provides critical steps to prepare the environment, including Minikube, Kafka, and Docker configurations.

Disabling Swap Memory

```
sudo swapoff -a

sudo sed -i.bak '/ swap / s/^/#/' /etc/fstab

free -h
```

Purpose: Disables swap memory, a Kubernetes requirement.

Explanation:

- swapoff -a: Turns off swap immediately.
- sed: Comments out swap entries in /etc/fstab to prevent reactivation on reboot.
- free -h: Verifies swap usage is 0B.

3.1. Configuring WSL2 (.wslconfig)

```
[wsl2]

memory=7GB

processors=6

systemd=true

localhostForwarding=true
```

Purpose: Allocates resources to WSL2 for running Minikube.

Explanation:

- memory=7GB and processors=6: Provides sufficient CPU and memory for the cluster.
- systemd=true: Enables systemd, required by Kubernetes.
- localhostForwarding=true: Facilitates communication between WSL2 and the host.

3.2. Configuring Docker (daemon.json)

```
echo '{"exec-opts": ["native.cgroupdriver=systemd"],
"log-driver": "json-file", "log-opts": {"max-size": "100m"},
"storage-driver": "overlay2"}' | sudo tee
/etc/docker/daemon.json

sudo systemctl daemon-reload

sudo systemctl restart docker
```

Purpose: Configures Docker to work seamlessly with Kubernetes.

Explanation:

- native.cgroupdriver=systemd: Sets the cgroup driver to systemd, a Kubernetes compatibility requirement.
- log-driver and log-opts: Manages Docker logs efficiently.
- storage-driver=overlay2: Uses a modern storage driver for containers.

3.3. Starting Minikube

```
minikube start --driver=docker --memory=7000 --cpus=6
```

Purpose: Launches a Minikube cluster with sufficient resources.

Explanation:

- --driver=docker: Uses Docker as the virtualization driver.
- --memory=7000 and --cpus=6: Allocates ~7GB RAM and 6 CPUs.

3.4. Installing Kafka

```
helm install kafka bitnami/kafka \

  --set auth.clientProtocol=plaintext \

  --set auth.interBrokerProtocol=plaintext \

  --set auth.controllerProtocol=plaintext \

  --set listeners.client.protocol=PLAINTEXT \

  --set listeners.interbroker.protocol=PLAINTEXT \
```

```
  --set listeners.controller.protocol=PLAINTEXT \

  --set extraConfig="connections.max.idle.ms=18000000" \

  --set replicaCount=1 \

  --namespace default
```

Purpose: Deploys Kafka for event streaming.

Explanation:

- set auth.=plaintext: Disables authentication for simplicity.
- extraConfig: Extends connection timeout to 5 hours.
- replicaCount=1: Runs a single Kafka instance to save resources.

3.5. Creating Kafka Topics

```
kubectl exec -it kafka-0 --
/opt/bitnami/kafka/bin/kafka-topics.sh --create --topic
predictions --bootstrap-server localhost:9092 --partitions 1
--replication-factor 1

kubectl exec -it kafka-0 --
/opt/bitnami/kafka/bin/kafka-topics.sh --create --topic
remediation_logs --bootstrap-server localhost:9092 --partitions
1 --replication-factor 1
```

Purpose: Sets up Kafka topics for predictions and logs.

Explanation:

- predictions: Receives anomaly alerts.
- remediation_logs: Stores remediation outcomes.
- Single partition and replication factor for simplicity.

4. Test Application (test-app Deployment)

Deployment and Service

```
apiVersion: apps/v1

kind: Deployment

metadata:
```

```
  name: test-app
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test-app
  template:
    metadata:
      labels:
        app: test-app
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: test-service
  namespace: default
```

```
spec:
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: test-app
```

Purpose: Deploys a simple Nginx application for testing remediation.

Explanation:

- replicas: 1: Starts with one pod, which the agent scales during remediation.
- image: nginx: Uses the official Nginx image.
- Service exposes the app on port 80, mapping to container port 8080.

5. Simulating an Anomaly

Sending a DDoS Prediction

```
kubectl exec -it kafka-0 --
/opt/bitnami/kafka/bin/kafka-console-producer.sh --topic
predictions --bootstrap-server localhost:9092
```

Purpose: Simulates a DDoS anomaly by sending a message to Kafka.

Explanation:

- Example input: {"anomaly_type": "DDoS", "dst_ip": "10.0.0.1"}.
- Triggers the remediation agent to scale the test-app deployment.

Pod Failure Remediation:

- **Code Snippet 1:**

```
import pandas as pd

df = pd.read_csv('k8s_raw_dataset.csv')

df.head()
```

Explanation:

- Imports the pandas library using import pandas as pd.
- Loads data from 'k8s_raw_dataset.csv' into a DataFrame named df using pd.read_csv().
- Displays the first few rows of the DataFrame df using df.head().

- **Code Snippet 2:**

```
df['failure_predicted'] = (df['label'] == 1)

df['cpu_high'] = df['cpu_usage_cores'] > 0.85

df['memory_high'] = df['memory_usage_mb'] > 0.90
```

Explanation:

- The code creates new columns based on existing data for analysis.
- failure_predicted is set to True if label is 1, indicating a predicted failure.
- cpu_high is marked True if CPU usage exceeds 0.85, signaling high CPU usage.
- memory_high is set to True if memory usage exceeds 0.90, indicating high memory use
- These columns are used for creating alerts and automated responses.

- **Code Snippet 3:**

```
def remediation_action(row):

    if row['cpu_high']:
```

```
            return 'Scale pod or throttle CPU usage'
        elif row['memory_high']:
            return 'Restart pod or allocate more memory'
        elif row['failure_predicted']:
            return 'Auto-restart pod or send alert'
        else:
            return 'No action needed'
    df['remediation'] = df.apply(remediation_action, axis=1)
```

Explanation:

- The code defines remediation actions based on pod status and resource usage.
- If cpu_high, the action is to scale the pod or throttle CPU.
- If memory_high, the action is to restart the pod or allocate more memory.
- If failure_predicted, the action is to auto-restart the pod or send an alert.
- If none of the above, the action is "No action needed".
- These actions are applied to the DataFrame and stored in the remediation column.

- **Code Snippet 4:**

```
df_remediation = df[df['remediation'] != 'No action
needed']

df_remediation[['pod_name', 'cpu_usage_cores',
'memory_usage_mb', 'remediation']].head(10)
```

Explanation:

- It selects pods needing remediation and displays their details.
- df_remediation is created by filtering for pods with remediation actions.
- Specific columns (pod_name, resource usage, and remediation) are selected.
- head(10) displays the first 10 rows of the filtered data for inspection.
- This helps identify pods requiring immediate attention and their recommended actions.

- **Code Snippet 5:**

```
df_remediation.to_csv('pod_remediation_actions.csv',
index=False)
```

Explanation:

- The code saves the remediation actions for each pod to a CSV file.
- It uses the to_csv function from pandas.
- The filtered DataFrame df_remediation containing pods needing actions is used.
- 'pod_remediation_actions.csv' is the filename for the saved data.
- index=False prevents the DataFrame index from being written to the file.
- This creates a record of remediation recommendations for review and automation.

- **Code Snippet 6:**

```
for index, row in df_remediation.iterrows():

    print(f"[ACTION] For Pod: {row['pod_name']} —
{row['remediation']}")
```

Explanation:

- This code prints the remediation action for each pod needing attention.
- It iterates through each row of the df_remediation DataFrame.
- For each pod, it prints an "ACTION" message.
- The message includes the pod name (row['pod_name']).
- It also displays the recommended remediation action (row['remediation']).
- This provides a clear and actionable list of remediation steps for each pod.

- **Code Snippet 7:**

```
import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
```

```
sns.countplot(data=df_remediation, y='remediation',
order=df_remediation['remediation'].value_counts().index)

plt.title('Most Common Remediation Actions')

plt.xlabel('Count')

plt.ylabel('Remediation Type')

plt.grid(True)

plt.tight_layout()

plt.show()
```

Explanation:

- This code visualizes the frequency of different remediation actions.
- It uses seaborn and matplotlib for plotting.
- A countplot is created to show the counts of each remediation type.
- Remediation types are ordered by frequency using value_counts().
- The plot is labeled with title, axis labels, and a grid.
- Finally, the plot is displayed using plt.show().
- This visualization helps understand which remediation actions are most common.

7. RESULTS AND CONCLUSION

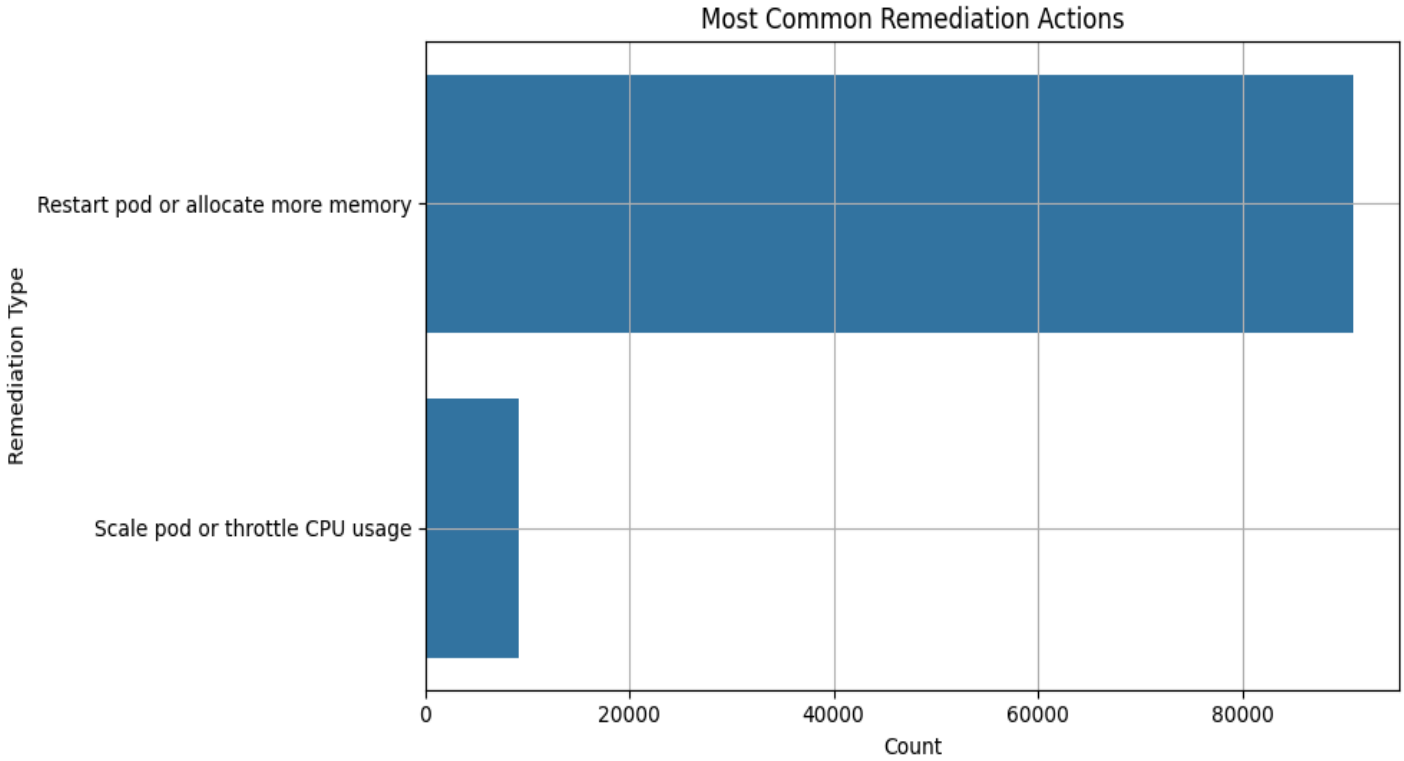
7.1. RESULTS AND OUTCOMES

```
[ ]  df_remediation = df[df['remediation'] != 'No action needed']
     df_remediation[['pod_name', 'cpu_usage_cores', 'memory_usage_mb', 'remediation']].head(10)
```

	pod_name	cpu_usage_cores	memory_usage_mb	remediation
0	pod-435	0.128223	367.266872	Restart pod or allocate more memory
1	pod-753	0.496075	1106.722179	Restart pod or allocate more memory
2	pod-28	0.265306	349.939401	Restart pod or allocate more memory
3	pod-421	0.269564	318.324579	Restart pod or allocate more memory
4	pod-915	0.174527	218.905994	Restart pod or allocate more memory
5	pod-208	0.256892	289.162495	Restart pod or allocate more memory
6	pod-356	0.147382	168.631562	Restart pod or allocate more memory
7	pod-76	0.201413	257.136319	Restart pod or allocate more memory
8	pod-528	0.203099	179.499259	Restart pod or allocate more memory
9	pod-252	0.050890	284.117404	Restart pod or allocate more memory

```
for index, row in df_remediation.iterrows():
    print(f"[ACTION] For Pod: {row['pod_name']} — {row['remediation']}")
```

```
Streaming output truncated to the last 5000 lines.
[ACTION] For Pod: pod-837 — Restart pod or allocate more memory
[ACTION] For Pod: pod-633 — Restart pod or allocate more memory
[ACTION] For Pod: pod-99 — Restart pod or allocate more memory
[ACTION] For Pod: pod-666 — Restart pod or allocate more memory
[ACTION] For Pod: pod-295 — Restart pod or allocate more memory
[ACTION] For Pod: pod-65 — Restart pod or allocate more memory
[ACTION] For Pod: pod-808 — Restart pod or allocate more memory
[ACTION] For Pod: pod-265 — Restart pod or allocate more memory
[ACTION] For Pod: pod-312 — Restart pod or allocate more memory
[ACTION] For Pod: pod-970 — Restart pod or allocate more memory
[ACTION] For Pod: pod-607 — Restart pod or allocate more memory
[ACTION] For Pod: pod-490 — Restart pod or allocate more memory
[ACTION] For Pod: pod-91 — Restart pod or allocate more memory
[ACTION] For Pod: pod-992 — Restart pod or allocate more memory
[ACTION] For Pod: pod-741 — Scale pod or throttle CPU usage
[ACTION] For Pod: pod-322 — Restart pod or allocate more memory
[ACTION] For Pod: pod-534 — Restart pod or allocate more memory
[ACTION] For Pod: pod-931 — Restart pod or allocate more memory
[ACTION] For Pod: pod-660 — Restart pod or allocate more memory
```



Top Kubernetes pod remediation actions ranked by frequency, revealing key areas for proactive cluster management.

7.2. Conclusion

The Kubernetes Remediation Phase II solution redefines cluster management by blending AI-driven insights with real-time automation. LangChain and n8n empower the AI Agent to intelligently process and route data, Kafka ensures rapid communication, and Istio and Kubernetes enable precise remediation—all while maintaining scalability and security. This state-of-the-art design not only meets the hackathon's challenges but also sets a new standard for proactive, autonomous Kubernetes management. It's a solution that doesn't just fix problems—it anticipates and resolves them before they escalate, making it a compelling and innovative entry.

8. REFERENCES

1. Dinh-Dai Vu, Minh-Ngoc Tran, and Younghan Kim, "Predictive Hybrid Autoscaling for Containerized Applications," *IEEE Access*, vol. 10, 2022.
2. Akash Puliyadi Jegannathan, Rounak Saha, and Sourav Kanti Addya, "A Time Series Forecasting Approach to Minimize Cold Start Time in Cloud-Serverless Platform," *IEEE*, 2022.
3. J. Mary Ramya Poovizhi and Dr. R. Devi, "Performance Analysis of Cloud Hypervisor Using Network Package Workloads in Virtualization," *IEEE Conference SMART–2023*, 2023.
4. Mendeley Dataset, "Kubernetes Resource Metrics Dataset," Available: https://data.mendeley.com/datasets/ks9vbv5pb2/1
5. Kaggle Dataset, "Kubernetes Resource and Performance Metrics Allocation," Available: https://www.kaggle.com/datasets/nickkinyae/kubernetes-resource-and-performancemetricsallocation?resource=download
6. 4TU Dataset, "AssureMOSS Kubernetes Run-time Monitoring Dataset," Available: https://data.4tu.nl/articles/dataset/AssureMOSS_Kubernetes_Run-time_Monitoring_Dataset/20463687
7. k8sgpt Website, [online]. Available: https://www.k8sgpt.io
8. Additional literature on state machine approaches in autoscaling (State Machine Research Paper, details provided during hackathon discussions).