GUIDEWIRE

Getting Started with KubeSight

Intelligent Kubernetes Failure Detection with AI-Driven Insights

GUIDEWIRE

KubeSight

Contents of this KubeSight

Introduction	Overview of KubeSight – a smart Kubernetes failure prediction and response system using ML & Gemini AI integration.
Proposed Solution	Real-time failure analysis, automated insights, and explainable AI responses integrated with Kubernetes metrics
WorkFlow/Technologies	Built using Flask, React + Vite, TailwindCSS, Prometheus, Grafana, ML models.
USP(Unique Feature)	Kubernetes Q&A chatbot, dynamic failure insights, and an interactive UI for engineers..
MVP(Screenshots/Demo Video)	Includes live dashboard, prediction form, and AI-powered advice sections – fully deployed on Vercel & Render.
Conclusion	KubeSight addresses critical infrastructure needs with ML automation and intuitive monitoring.

GUIDEWIRE

KubeSight

01 Introduction

What is KubeSight?
Why Kubernetes monitoring is critical
Existing gaps in failure prediction and response

Let’s have a look

KubeSight is an AI-driven platform that forecasts failures in Kubernetes clusters and offers intelligent, context-aware suggestions. It blends machine learning to automatically identify problems and recommend solutions—reducing downtime and enhancing system reliability.

Why Kubernetes Monitoring is Essential?

Kubernetes is a dynamic, constantly shifting system of workloads moving in and out of pods and nodes. Real-time visibility becomes not just useful, but necessary.

KubeSight solves this problem by providing a predictive intelligence layer on top of the current monitoring stack. Rather than basing decisions on hard-coded thresholds, it applies the judgment of a trained machine learning model to examining real-time metrics—like CPU usage, memory pressure, and network I/O—and recognizing patterns that presage possible failure.

This enables it to look past reactive alerting by forecasting which problems are likely to occur and why, which provides users an important head start.

GUIDEWIRE

KubeSight

02 Proposed Solution

Turning the Vision of KubeSight into Reality

GUIDEWIRE

KubeSight

Proposed Solution

The first vision for KubeSight was to build an intelligent and proactive solution for Kubernetes failure prediction and fixing. Seeing the increasing complexity in operating Kubernetes clusters, our vision was to go beyond the conventional dashboards and develop a system that not only identifies problems in real time but also predicts them with machine learning and offers intelligent fixes through a conversational interface.

Work done in Phase 1:

- We started gathering real time data from Prometheus using codes to extract and simulated Kubernetes metrics focusing on CPU,memory,network i/o,pod status and node health.
- Then the next step was to set up ML pipeline with synthetic +real-world metrices for that we built a RandomForestClassifier to predict Pod/node failures/CPU/memory/resource exhaustion/network issues/service disruptions chosen for real-time readiness.
- Then used 80/20 train-test split dataset and high accuracy and balanced F1-score and evaluated with confusion matrix and visualize the dataset using graphs.
- Prepared backend with Flask API and structured code for Helm-based Kubernetes deployment in Phase 2.

Continue ...

Work Done in Phase 2 – Remediation & Automation (Aligned with Guidewire Objectives)

Following the successful implementation of an ML-based prediction model in Phase 1, **Phase 2** focused on transforming predictions into **real-time, actionable remediation**—making **KubeSight** a complete intelligent agent for Kubernetes reliability. This phase emphasized building a system that could not only detect issues early but also provide smart responses or recommendations to mitigate them.

Remediation System Design

- Using the output from the prediction engine (e.g., “High Memory Usage” or “Pod Failure Likely”), we implemented a rule-based response system that generates **suggestions**. Each class of failure was mapped to remediation actions based on DevOps best practices
- **Predicted Resource Exhaustion (CPU/Memory):**
- Recommended scaling pods or increasing resource limits.
- **Forecasted Pod/Node Failures:**
  → Suggested restarting, rescheduling, or relocating pods.
- **Network or Service Bottlenecks:**
  → Offered advice to check service dependencies, DNS, or connection policies.
- These recommendations were shown on the **Solution Page** after a prediction and also in a separate **Gemini-powered Q&A interface**, allowing users to interactively query and understand each issue.

- User testing the sequence: **Prediction → Explanation → Resolution Suggestion**.
- Although full deployment in Kubernetes was optional, we prepared the backend as a containerized **Flask service**, with all dependencies wrapped using Docker. The code is structured and commented for easy Helm chart generation and Kubernetes deployment in real clusters.

GUIDEWIRE

KubeSight

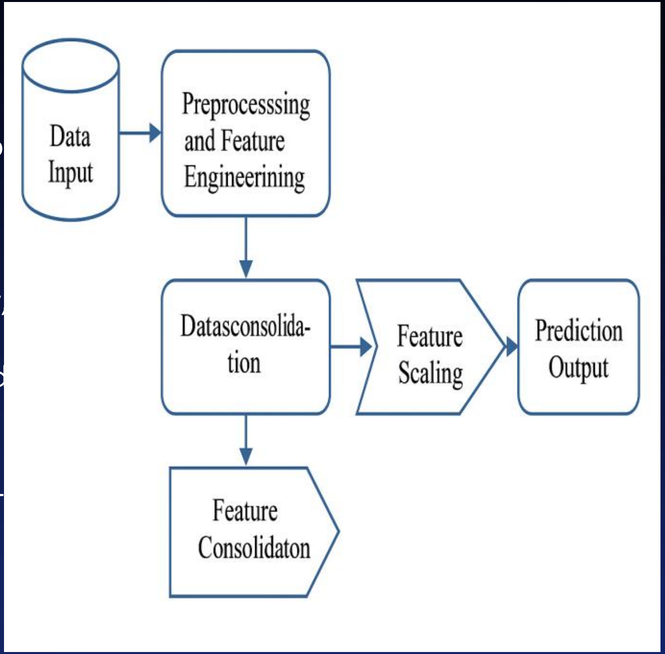
03

WorkFlow/Technologies

Turning the Vision of KubeSight into Reality

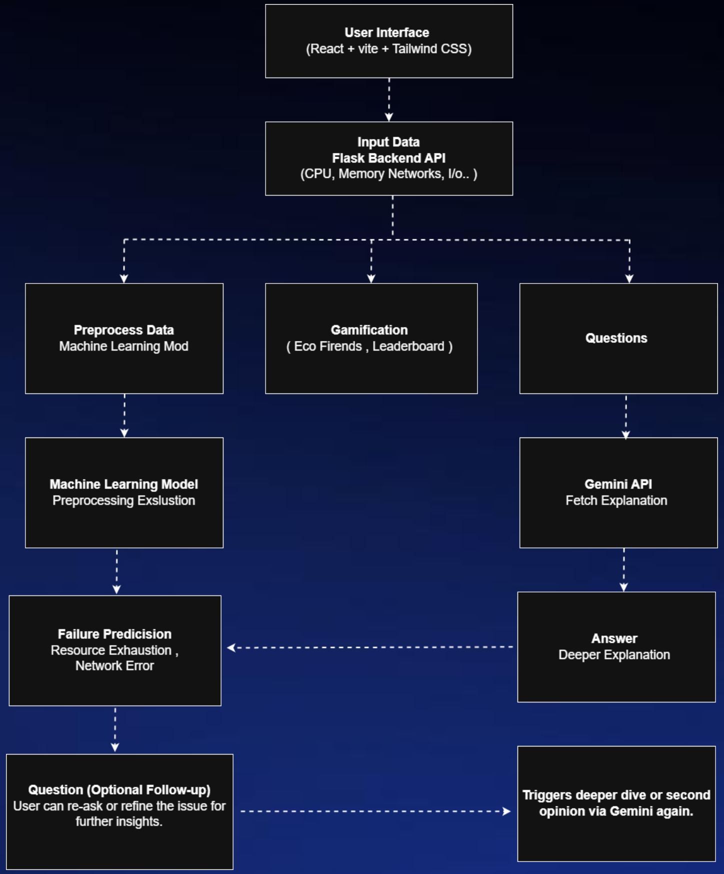
Phase 1 Workflow – Prediction System Development

- **Problem Identification-** Recognize the need for predicting Kubernetes Failure.
- **Data** Collection-Gather the stimulated data metrics in the csv form that includes CPU usage, pod names, memory usage, pod server, pod status , network I/O from Prometheus , public datasets and synthetic Datasets.
- **Data Preprocessing-** Clean and normalize the data so remove null valuses and or to deal with non integer data and structure it into a format suitable for model training
- **Feature Selection-** Select key features that are needed to test the Model or those that are needed to be considered for pod failures.
- **Model Selection-** We chose **RandomForestClassifier** due to its high interpretability, low training time, and robustness for tabular datasets.
- **Training the Model-** Trained the model on 80% of the dataset using scikit-learn and optimized the predicted issues like pod failures, resource exhaustion, and network issues.
- **Evaluation & Testing-** Tested on the remaining 20% using metrics like accuracy, F1-score, and confusion matrix to ensure reliable failure classification.
- **Model Export & API Setup- S**aved the trained model with joblib and encapsulated it within Flast API endpoint(/predict) to deliver predictions in real-time.
- **Ready for Phase 2**



Phase 2 Workflow – Remediation & Automation System

- Problem Analysis- Analyse how the predicted failure can be fixed in Kubernetes and how can customers solve their problems by own with few guidance.
  Remediation Mapping- Translated every anticipated problem (for instance, CPU overload, pod crash, network outage) into respective remediation approaches as per DevOps best practices.
- Gemini AI Integration- Integrated Gemini to give step by step explanation of the steps to resolve the error and it can also resolves user questions regarding Kubernetes.
- Backend Logic Expansion-Extended the Flask backend with a new endpoint (/ask-advice) to process anticipated problems and dynamically retrieve customized responses from Gemini.
- UI/UX Enhancement- Created a page For Prediction where after analyzing the model will give the predictions and the proposed solution and explanation that can be understood by human .
- Effectiveness Testing- Monitored relevance and comprehension of Gemini recommendations, and ensured them with specified test cases as well as in actual scenarios.
- Pre-Deployment Setup- Containerized backend and frontend, with a code structure which supports Helm chart deployment and also Kubernetes compatibility.
- Ultimate Integration- Integrated prediction + remediation into a seamless, real-time user experience where users enter metrics → receive prediction → see actionable recommendations in real-time.



User Interface
(React + vite + Tailwind CSS)

Input Data
Flask Backend API
(CPU, Memory Networks, I/o..)

Preprocess Data
Machine Learning Mod

Gamification
(Eco Firends , Leaderboard)

Questions

Machine Learning Model
Preprocessing Exslustion

Gemini API
Fetch Explanation

Failure Predicision
Resource Exhaustion ,
Network Error

Answer
Deeper Explanation

Question (Optional Follow-up)
User can re-ask or refine the issue for further insights.

Triggers deeper dive or second opinion via Gemini again.

Technology

User Interface

- Collects Kubernetes-related inputs (like CPU, memory, I/O) from the user.
- Provides a clean interface for real-time monitoring and interaction.

Backend Server (Flask)

- Acts as the bridge between frontend and ML pipeline.
- Processes HTTP requests and passes data to preprocessing.

Preprocessing (Two Approaches Demonstrated)

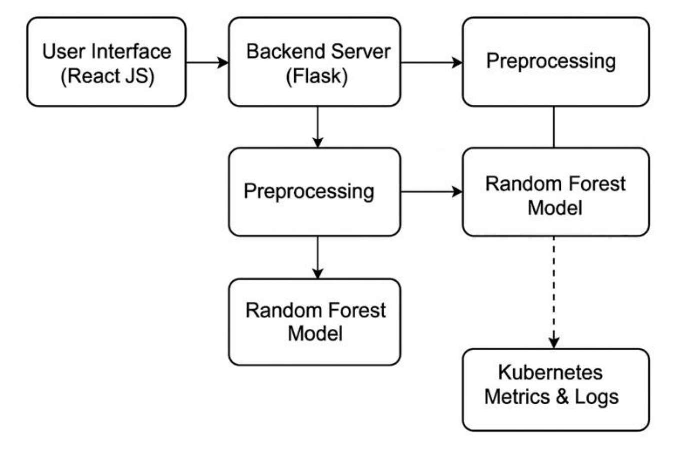
- Cleans and formats the incoming raw metric data.
- Handling missing values, scaling, and encoding if necessary.
- Feeds the same cleaned data to a separate Random Forest model for validation/testing (modular flexibility).

Main Random Forest Block:

- It uses preprocessed data to make predictions of failures.
- Output predictions such as: "Excessive CPU use", "Memory spike", etc.

Side Random Forest Block:

- Could represent either a backup model or real-time ensemble evaluation.



04 USP(Unique Feature)

Unique Features in KubeSight

USP

- KubeSight not just monitor the issues with Kubernetes and predict it before it happens but it also helps them prevent it with detailed and human understandable words.
- KubeSight integrates with Gemini to provide contextual, Kubernetes-specific suggestions—ranging from scaling recommendations to network repairs—all in natural language.
- The “More Info” option will take you to the Question/Answer page where the user can ask their doubts and can guide them step by step throughout the issue.
- Created a well interactive UI which will make things understandable how and what will cause the Kubernetes failure.

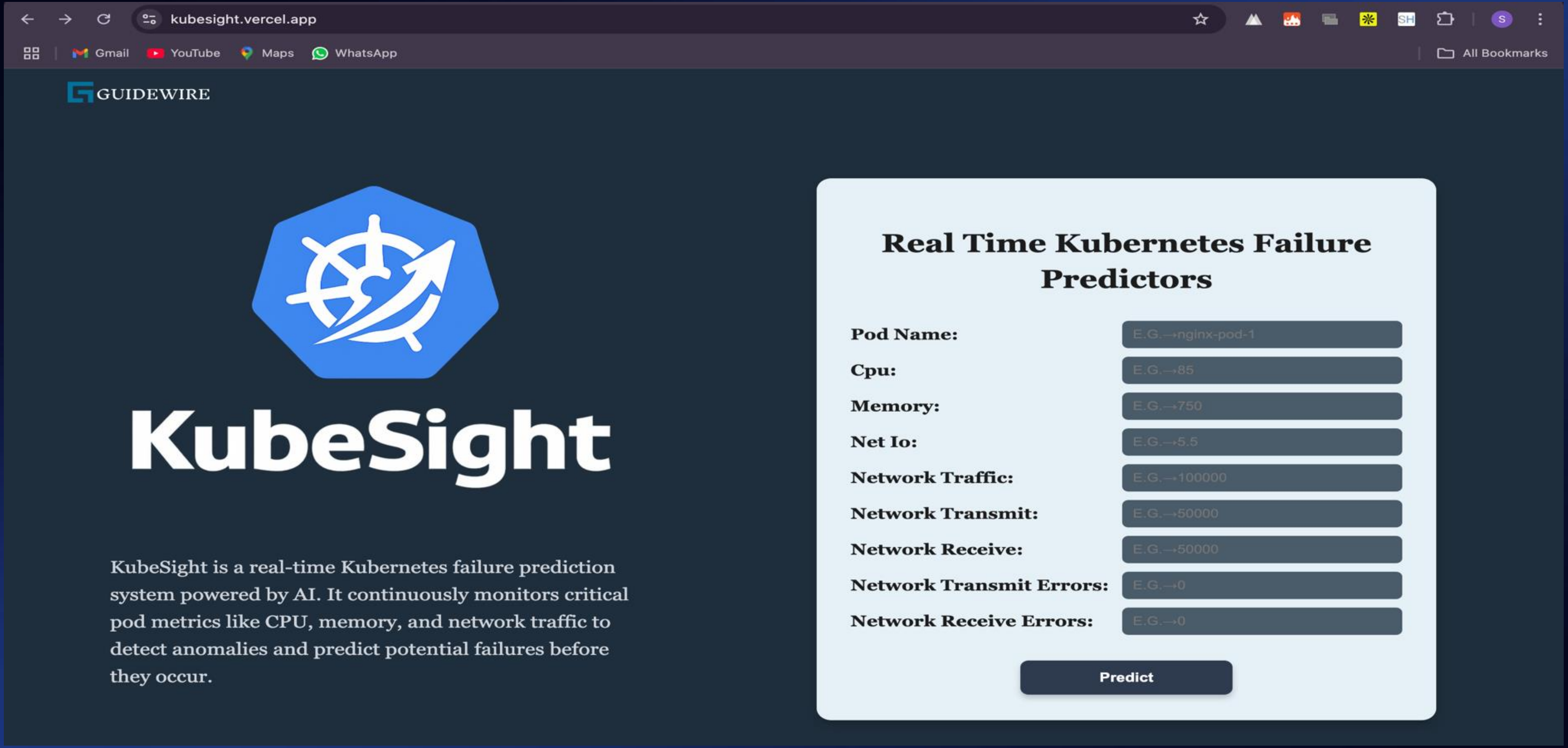
05 MVP (Screenshots/Demo Video)

GUIDEWIRE

KubeSight

Screenshots

Home/Predictor Page



KubeSight

KubeSight is a real-time Kubernetes failure prediction system powered by AI. It continuously monitors critical pod metrics like CPU, memory, and network traffic to detect anomalies and predict potential failures before they occur.

Real Time Kubernetes Failure Predictors

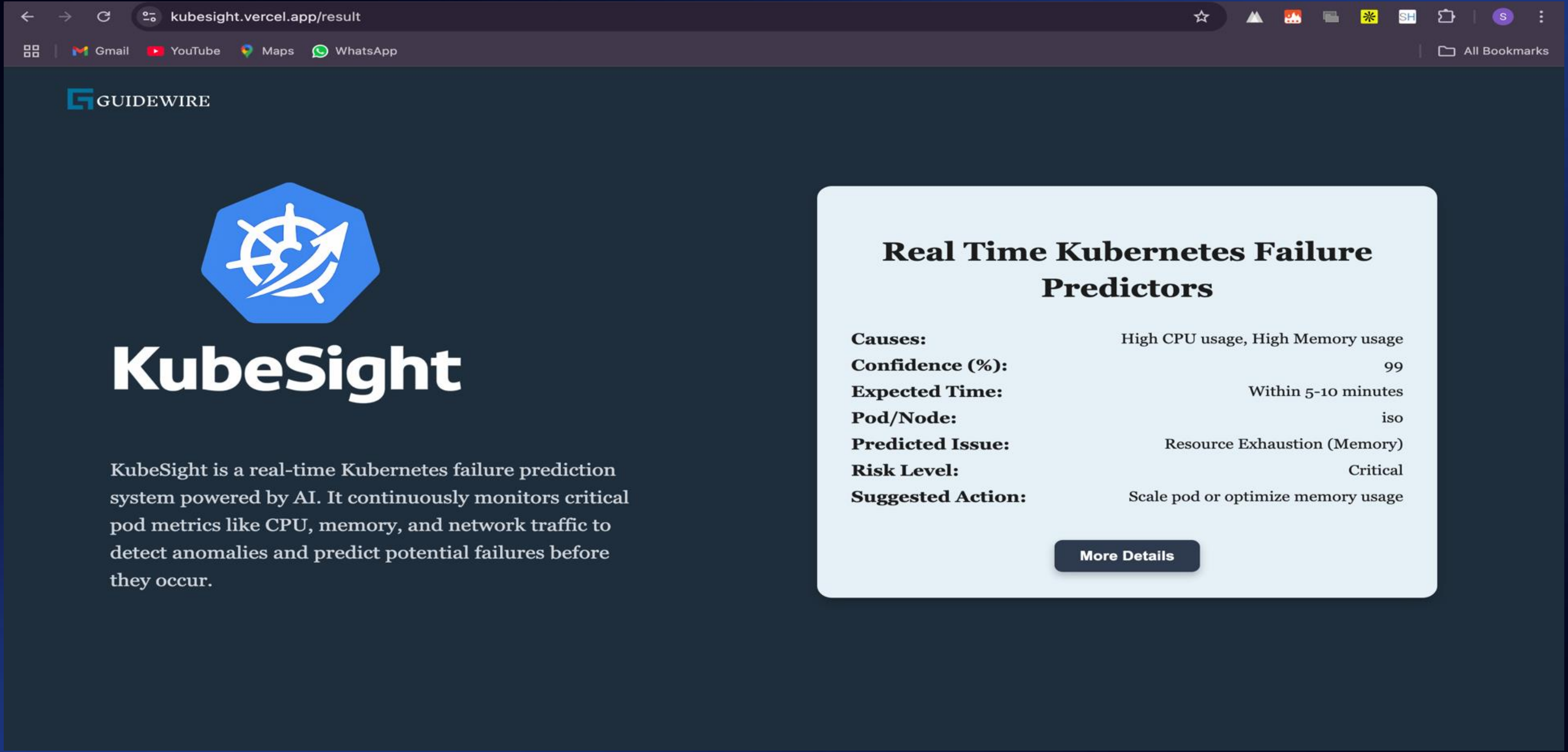
Pod Name:	E.G.→nginx-pod-1
Cpu:	E.G.→85
Memory:	E.G.→750
Net Io:	E.G.→5.5
Network Traffic:	E.G.→100000
Network Transmit:	E.G.→50000
Network Receive:	E.G.→50000
Network Transmit Errors:	E.G.→0
Network Receive Errors:	E.G.→0

GUIDEWIRE

KubeSight

Screenshots

Home/Result Page:



KubeSight is a real-time Kubernetes failure prediction system powered by AI. It continuously monitors critical pod metrics like CPU, memory, and network traffic to detect anomalies and predict potential failures before they occur.

Real Time Kubernetes Failure Predictors

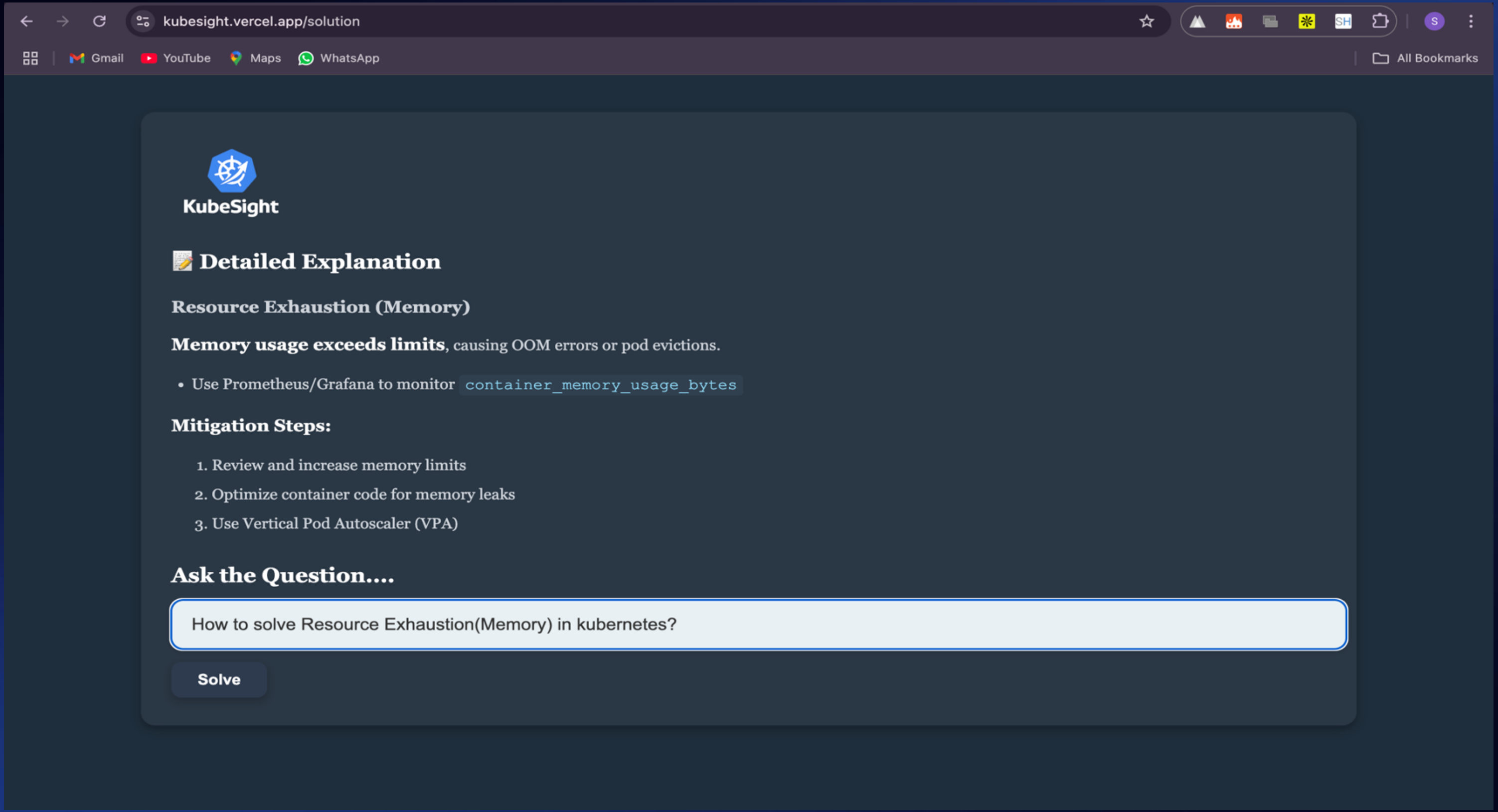
Causes:	High CPU usage, High Memory usage
Confidence (%):	99
Expected Time:	Within 5-10 minutes
Pod/Node:	iso
Predicted Issue:	Resource Exhaustion (Memory)
Risk Level:	Critical
Suggested Action:	Scale pod or optimize memory usage

GUIDEWIRE

KubeSight

Screenshots

Home/Result Page:



📝 Detailed Explanation

Resource Exhaustion (Memory)

Memory usage exceeds limits, causing OOM errors or pod evictions.

- Use Prometheus/Grafana to monitor `container_memory_usage_bytes`

Mitigation Steps:

1. Review and increase memory limits
2. Optimize container code for memory leaks
3. Use Vertical Pod Autoscaler (VPA)

Ask the Question....

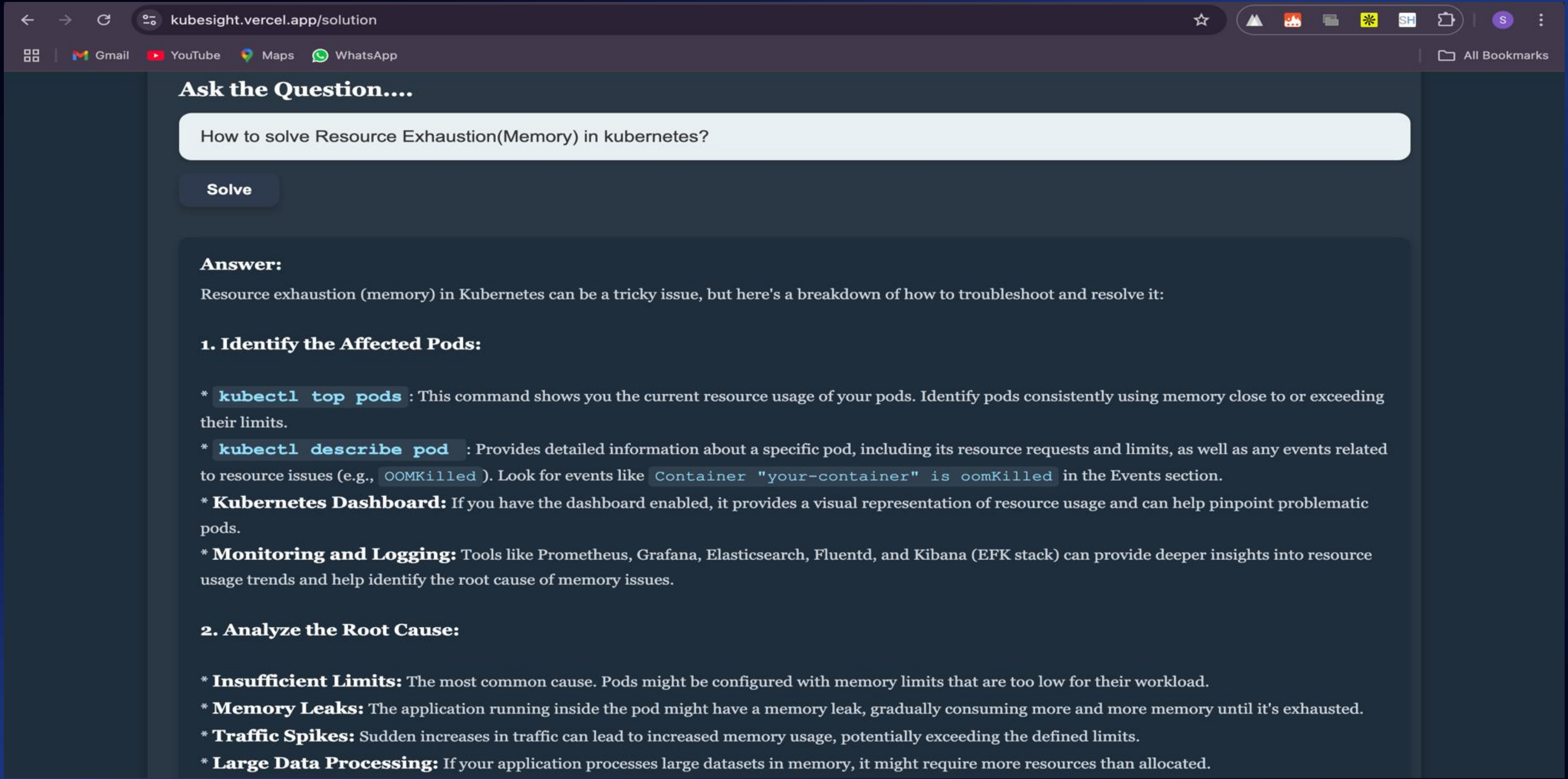
How to solve Resource Exhaustion(Memory) in kubernetes?

GUIDEWIRE

KubeSight

Continue

Home/Response Page:



Ask the Question....

How to solve Resource Exhaustion(Memory) in kubernetes?

Answer:

Resource exhaustion (memory) in Kubernetes can be a tricky issue, but here's a breakdown of how to troubleshoot and resolve it:

1. Identify the Affected Pods:

* `kubectl top pods` : This command shows you the current resource usage of your pods. Identify pods consistently using memory close to or exceeding their limits.

* `kubectl describe pod` : Provides detailed information about a specific pod, including its resource requests and limits, as well as any events related to resource issues (e.g., `OOMKilled`). Look for events like `Container "your-container" is oomKilled` in the Events section.

* **Kubernetes Dashboard:** If you have the dashboard enabled, it provides a visual representation of resource usage and can help pinpoint problematic pods.

* **Monitoring and Logging:** Tools like Prometheus, Grafana, Elasticsearch, Fluentd, and Kibana (EFK stack) can provide deeper insights into resource usage trends and help identify the root cause of memory issues.

2. Analyze the Root Cause:

* **Insufficient Limits:** The most common cause. Pods might be configured with memory limits that are too low for their workload.

* **Memory Leaks:** The application running inside the pod might have a memory leak, gradually consuming more and more memory until it's exhausted.

* **Traffic Spikes:** Sudden increases in traffic can lead to increased memory usage, potentially exceeding the defined limits.

* **Large Data Processing:** If your application processes large datasets in memory, it might require more resources than allocated.

GUIDEWIRE

KubeSight

Demo Part

Demo Video - Video
GitHub Link - Link
Website - https://kubesight.vercel.app/

06 Conclusion

KubeSight summary

Conclusion

KubeSight is an artificial intelligence integarted Kubernetes failure predictor and solution recommendation system that's used to improve cluster management by being faster, smarter and useful. It integrates a machine learning algorithm that has been trained on real and simulated cluster metrics to forecast probable failures and propose smart, context-aware recommendations. The solution includes a trendy React-based user interface, a Flask backend delivering real-time forecasts, and a conversational UI for troubleshooting. Whether it's resource starvation, pod crash, or network problems—KubeSight not only alerts them early but also informs you how to repair them.

KubeSight closes the observability-action gap by converting raw metrics into actionable insights—enabling DevOps teams to minimize downtime, respond quickly, and keep clusters healthier with ease.

GUIDEWIRE

KubeSight

Our Teams

Shreya Singh

shreya131268@gmail.com

Github || LinkedIn

Prince Kumar

princebaghal637@gmail.co

Github |m| LinkedIn

Harshit Gola

golaharsh0411@gmail.com

Github || LinkedIn

GUIDEWIRE

KubeSight

Thank you