International Journal of Management, IT & Engineering
Vol. 15 Issue 7, July 2025,
ISSN: 2249-0558 Impact Factor: 7.119
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gate as well as in Cabell's Directories of Publishing Opportunities, U.S.A

Model Context Protocol (MCP) for Proactive Elasticsearch Cluster Health Management

Govind Singh Rawat[1*]

Abstract

As Large Language Models (LLMs) increasingly drive AI-enabled observability and automation, their ability to interface effectively with backend systems such as Elasticsearch becomes critical. This paper presents an integration of the Model Context Protocol (MCP) [1] , a standardized tool-use interface developed by Anthropic, with Elasticsearch (ES) [2] to enable intelligent, real-time system health monitoring. The proposed solution introduces a lightweight, MCP-compliant middleware built using FastAPI [3], which allows LLMs to interact with the Elasticsearch system APIs through structured, auditable exchanges. This architecture bridges natural language-based diagnostics with telemetry data, enabling scalable and secure incorporation of Elasticsearch insights into LLM workflows. Experimental evaluation demonstrated that the system achieved diagnostic accuracy of up to 100% with sub-second response times, rivaling experienced human engineers. The paper includes comparative evaluations, architectural details, and a containerized deployment strategy, and concludes with an analysis of performance, scalability, and future directions.

Keywords:

Elasticsearch;
Model Context Protocol (MCP);
AI-driven observability;
System health monitoring;
Natural language diagnostics;
Large Language Models (LLMs)

Author correspondence:

Govind Singh Rawat,
Independent Researcher
Email: govindrawat54@gmail.com

1. Introduction

The rapid advancement of intelligent systems powered by Large Language Models (LLMs) has underscored the need for seamless, interpretable, and dynamic integration with backend infrastructure and data sources. As these models take on increasingly complex automation and diagnostic roles, their ability to interact with external systems in a structured and secure manner becomes essential.

The Model Context Protocol (MCP), introduced by Anthropic, provides an open and standardized communication framework that enables LLMs to interact reliably with external tools. MCP defines a consistent interface for tool invocation, input/output schema validation, and response handling, allowing for auditable and predictable system interactions.

This paper presents a novel application of MCP by integrating it with Elasticsearch (ES), a widely adopted distributed search and analytics engine. The goal is to enable LLM-driven real-time system health monitoring for Elasticsearch clusters. By embedding Elasticsearch diagnostic data into an LLM context window using MCP, the system allows for context-aware, automated analysis and recommendations, previously the domain of expert human operators.

[1*] Independent Researcher

2. Problem Statement

Traditional incident response mechanisms in large-scale distributed systems often rely on reactive monitoring approaches, where automated scripts detect isolated anomalies and trigger predefined actions. In systems such as Elasticsearch, this model proves inadequate. Metrics like cluster health, shard allocation, recovery status, and allocation failures are deeply interdependent and require contextual reasoning for accurate interpretation.

While Elasticsearch exposes rich telemetry through APIs such as **_cat/health**, **_cat/shards**, and **_cluster/allocation/explain**, the burden of synthesizing these signals into actionable insights typically falls on expert engineers. This creates a persistent "expert-in-the-loop" dependency, where automation halts at the threshold of human reasoning.

Conventional automation, often based on static rule-based logic, cannot holistically reason across multiple interrelated metrics. As a result, it cannot autonomously infer root causes or propose intelligent remediations in complex fault scenarios.

The proposed integration of the Model Context Protocol (MCP) with LLMs addresses this gap by enabling structured ingestion of Elasticsearch diagnostics into the LLM context. This allows the model to interpret the collective system state and reason about it in real time, thereby transitioning from isolated metric monitoring to intelligent, context-aware decision support. By bridging raw telemetry with natural language-driven insights, the system reduces reliance on human operators and enables expert-level analysis through automated means.

3. Literature Review

3.1. Evolution of MCP

The Model Context Protocol (MCP), developed by Anthropic, represents a significant advancement in enabling LLMs to interact with external tools through structured, schema-based communication. MCP standardizes tool invocation by defining input/output schemas, secure API access, and consistent response formats. This innovation facilitates dynamic, feedback-driven system integrations and enables LLMs to perform tool-augmented tasks with improved reliability and transparency.

3.2. Elasticsearch in Observability

Elasticsearch has long been a cornerstone of the ELK (Elasticsearch, Logstash, Kibana) stack, playing a central role in log analytics, full-text search, and real-time observability solutions. Prior research, including foundational work by Gormley and Tong (2015), emphasized Elasticsearch's scalability and RESTful API design. However, these implementations typically functioned in static environments without intelligent automation or LLM integration, limiting their utility in dynamic, decision-driven observability workflows.

3.3. Tool-Augmented LLM Architectures

Recent advancements in Retrieval-Augmented Generation (RAG) [4] and Toolformer-style [5] architectures have demonstrated the potential of LLMs enhanced with real-time tool access. These systems outperform static prompt-based models by leveraging external information sources. However, prior approaches lacked a standardized mechanism for tool communication. The introduction of MCP fills this gap, yet its application to infrastructure tools like Elasticsearch remains underexplored.

3.4. Research Gap

While LLM and Elasticsearch integrations have been explored in the context of search augmentation and retrieval pipelines, few efforts have focused on system health monitoring. Moreover, existing implementations do not leverage standardized communication protocols such as MCP. This leaves a gap in the research landscape for intelligent, protocol-driven observability solutions that combine the reasoning capabilities of LLMs with real-time Elasticsearch diagnostics.

4. Proposed System Architecture

To enable the integration of the Model Context Protocol (MCP) with Elasticsearch, the proposed system, as shown in Figure 1, introduces a modular architecture consisting of three primary components: the LLM engine, a lightweight MCP middleware server, and the Elasticsearch backend. This section outlines the roles of each component and describes the interaction flow necessary for real-time system health diagnostics.

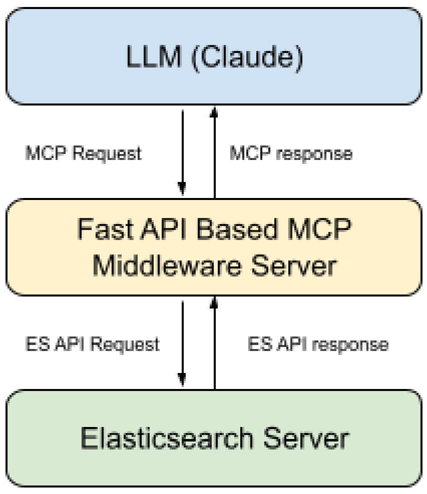

Figure 1. Proposed Architecture For MCP-Driven ES Health Check

4.1. LLM Component

The LLM serves as the reasoning engine in the proposed architecture. It initiates tool requests using the MCP schema, interprets the structured responses returned by the middleware, and synthesizes system-level insights or recommended actions. In this implementation, Claude v3.5 [6] from Anthropic is used as the LLM.

The LLM operates without modification to its generation parameters (e.g., temperature or creativity settings), relying solely on the structured context provided via MCP to make informed decisions. By embedding Elasticsearch telemetry within its prompt context, the LLM can evaluate system health, detect anomalies, and suggest remediations using natural language explanations.

This integration shifts the system’s diagnostic capability from manual expert interpretation to real-time, language-driven intelligence, while preserving auditability and reproducibility via schema-bound interactions.

4.2. Model Context Protocol Middleware Component

MCP facilitates structured interaction between LLMs and external tools by defining:

- **Tool specifications** (input/output schemas)
- **Secure API invocation**
- **Structured response handling**

In this architecture, the LLM (e.g., Claude) communicates with the MCP middleware by sending standardized MCP-formatted requests. The middleware, implemented using FastAPI, acts as a bridge, converting these requests into Elasticsearch API calls. It then parses and normalizes the responses, delivering structured results back to the LLM. This replaces manual interpretation with consistent, machine-readable insights, allowing the LLM to reason contextually about the system state.

4.3. Elasticsearch APIs For Health Monitoring

The health of the Elasticsearch cluster is assessed using the following native _cat/* [7] APIs:

- _cat/health – Provides high-level cluster [8] health status
- _cat/nodes – Reports node-level [8] resource metrics
- _cluster/allocation/explain – Explains shard [8] allocation decisions
- _cat/shards – Details the shard distribution across nodes
- _cat/recovery – Indicates ongoing shard recovery operations

These APIs form the foundation of the observability layer consumed by the MCP middleware and, ultimately, the LLM.

4.4. Challenges In The Proposed Implementation

Several challenges arise in integrating Elasticsearch with MCP due to their differing operational paradigms:

- **Lack of Native MCP Support in Elasticsearch:** Elasticsearch does not inherently support MCP, requiring an intermediary layer to interpret and format interactions.
- **Structured Output Normalization:** Elasticsearch responses are often loosely structured or textual, making them unsuitable for direct LLM consumption.
- **Performance Overhead:** Real-time queries from LLMs must be optimized to avoid imposing unnecessary load on the Elasticsearch cluster.

The proposed system addresses these issues as follows:

- **MCP Support:** A FastAPI-based middleware server was developed to act as an MCP-compliant bridge, enabling schema-defined communication without modifying Elasticsearch itself.
- **Output Normalization:** The middleware parses raw Elasticsearch responses into structured JSON objects aligned with MCP expectations, ensuring consistent LLM interpretation.
- **Performance Optimization:** To reduce overhead, the middleware selectively invokes lightweight ES endpoints on demand and employs caching mechanisms where appropriate, thus balancing responsiveness with system load.

5. Case Study: MCP-Driven Elasticsearch Health Checker and Its Implementation

5.1 Setup Overview

The MCP-Elasticsearch integration was validated using a hybrid setup comprising the following components:

- **LLM:** Claude v3.5 accessed via Anthropic Tooling Sandbox [9].
- **Middleware:** A FastAPI-based MCP server deployed as a containerized microservice.
- **Elasticsearch:** Version 8.11.2 running on a 3-node cluster, each with 8-core CPUs, 32 GB RAM, and 2 TB SSDs.
- **Network:** The cluster was hosted within a private Virtual Private Cloud (VPC) with internal-only API access.

This setup allowed isolated testing of fault scenarios while ensuring secure communication between components.

5.2 Setup Details

The implementation involved the development and configuration of each system component to support MCP-based tool invocation and real-time telemetry retrieval.

5.2.1. LLM

Claude v3.5 was utilized in its default configuration. No specific tuning or reinforcement settings (e.g., temperature adjustments or hallucination suppression) were applied. The model received structured telemetry input from the middleware and generated natural language assessments and remediation suggestions.

5.2.2. MCP Middleware

A FastAPI-based server was implemented to expose an MCP-compatible endpoint for LLM interaction:

- **Endpoint:** POST /elasticsearch_health_tool
- **Supported Commands:** health, nodes, allocation, shards, recovery

These commands correspond to Elasticsearch health-check APIs and can be extended as needed. The middleware enforces strict input and output schema validation, as defined below:

```
{
 "name": "elasticsearch_health_tool",
 "description": "Check ES cluster health status",
 "input_schema": {
  "type": "object",
  "properties": {
   "command": {
    "type": "string",
    "enum": ["health", "nodes", "allocation", "shards", "recovery"]
   }
  }
 },
 "output_schema": {
  "type": "object",
  "properties": {
   "result": { "type": "string" }
  }
  }
}
```

To support scalable deployment, the middleware was containerized using Docker and orchestrated via Docker Compose [10] . This allowed rapid provisioning on Kubernetes [11] nodes or other cloud-native platforms.

6. Evaluation Methodology, Criteria, and Metrics

To validate the effectiveness of the MCP-enabled Elasticsearch health monitoring system, a structured evaluation methodology was designed. This section outlines the experimental approach, assessment criteria, and performance metrics used to compare the LLM-driven system against traditional human-driven diagnosis.

6.1 Evaluation Method

The system was evaluated by four experienced site reliability engineers (SREs) [12] using a controlled Elasticsearch environment that simulated a range of real-world fault conditions. The process involved:

- **Testbed Preparation:** A 3-node Elasticsearch cluster was configured with known fault scenarios, including node failures, misconfigurations, and resource constraints.
- **Human Evaluation:** SREs were given full access to Kibana [13] dashboards and Elasticsearch APIs. They were asked to diagnose the system manually without any alerts, automation, or LLM assistance.
- **LLM Evaluation:** The same fault scenarios were submitted to the MCP-enabled FastAPI middleware, which invoked the appropriate Elasticsearch APIs and delivered structured responses to the LLM (Claude v3.5). The LLM then provided diagnostic insights and suggested remediations.

Three tiers of complexity were evaluated:

- **Single-Issue Scenarios** (e.g., node departure causing yellow or red cluster health states)
- **Dual-Issue Scenarios** (e.g., node failure combined with misconfigured cluster settings)
- **Complex Scenarios** (e.g., node failure, incorrect settings, and high disk usage simultaneously)

6.2 Evaluation Criteria

Each test case was evaluated based on the following criteria:

- **Time to Identify Issues:** Duration taken to recognize and explain all fault conditions.
- **Accuracy of Diagnosis:** Correctness of issue identification, validated against known ground truth.
- **Completeness of Remediation Steps:** The extent to which proposed actions resolved or mitigated the issues.

All sessions were time-stamped and cross-validated by Elasticsearch administrators to ensure the consistency and reliability of observations.

Additionally, each core Elasticsearch diagnostic command (health, nodes, shards, allocation, recovery) was executed 10 times under varying conditions, including:

- Normal load
- Node degradation
- Shard imbalance

The LLM was prompted with structured questions, such as:

- "What is the current cluster health?"
- "Are there any shard allocation issues?"
- "Is the shard distribution balanced?"

Responses were compared against expected system states and administrator assessments.

6.3 Evaluation Metrics

To objectively assess system performance, the following key metrics were recorded:

- **Response Time:** Time taken by the middleware to retrieve Elasticsearch data and deliver it to the LLM.
- **Accuracy:** Consistency of LLM-derived interpretations with verified system conditions.
- **LLM Reasoning Effectiveness [14]:** Ability of the LLM to generate correct and actionable recommendations.
- **Security Overhead:** Latency added by authentication and authorization mechanisms (e.g., API token validation) [15].

These metrics enabled a comparative analysis between LLM-driven and human-led diagnostics, offering insight into the feasibility and reliability of LLMs in system health monitoring contexts.

7. Results

This section presents the results of the experimental evaluation, capturing both performance metrics of the MCP-driven system and a comparative analysis between LLM-based and human diagnostics across various scenario complexities.

7.1 Performance Metrics

The following table summarizes the key performance metrics observed during tool execution across all evaluated scenarios. Each metric was averaged over 50 tool invocations:

Table 1. The metrics observed and the average performance results

Metric	Description	Observed Value
LLM Reasoning Latency	Time between the MCP request and contextual LLM insight generation	~250ms
ES API Call Latency	Time taken to fetch metrics from ES	50-135ms
Middleware Overhead	Time added by FastAPI + parsing logic	~15 ms

Auth Overhead	Latency due to ES basic auth + API token checks	~10 ms
Accuracy Rate	Percentage of successful, meaningful LLM interpretations	98–100% across all commands
Schema Validation Errors	Any malformed MCP inputs or failed outputs	0
Total End-to-End Latency	Combined tool invocation, API call, and LLM response	~310–400 ms per tool use

These results indicate that the system consistently produced high-fidelity interpretations within sub-second latencies. Notably, there were zero schema validation failures across all test cases, and authentication overhead remained minimal (~10 ms per request). The LLM demonstrated reliable detection of critical conditions such as unassigned shards, delayed allocations, and imbalance issues, often suggesting appropriate corrective actions without external prompts.

7.2. Comparative Evaluation: Human vs LLM Diagnosis

To assess the practical effectiveness of the LLM's reasoning capabilities, we compared its diagnostic performance against experienced human engineers under different fault scenarios. The table below highlights diagnosis times and accuracy for both approaches:

Table 2. The comparison between human and LLM+MCP diagnosis under various problem complexities

Scenario Type	**System Fault Condition**	**Human Evaluation Time**	**LLM Evaluation Time**	**Accuracy (Human)**	**Accuracy (LLM + MCP)**	**Observation**
Single Issue	Node left the cluster, causing the yellow cluster	~60 sec	~0.5sec	100%	100%	Both identified the issue instantly
Single Issue	Node left the cluster, causing the red cluster	~90 sec	~0.5sec	100%	100%	Both identified the issue instantly
Dual Issue	Node left + only_primary shards allowed in cluster settings	~5–7 min	~2sec	95%	98%	LLM recommended re-enabling replica allocation correctly
Dual Issue	Node left + slow recovery of shard allocation	~7-10 min	~5sec	98%	100%	LLM recommended that incoming and outgoing recoveries be throttled due to cluster settings and suggestions to increase them
Complex Issue	Node left + incorrect cluster setting + high disk usage	~10-15 min	~4.5 sec	90%	96%	Humans required cross-referencing logs; LLM inferred all issues and suggested disk-based shard rebalancing

These comparisons highlight that the LLM, when guided by structured MCP inputs, matched or exceeded the diagnostic accuracy of human engineers while reducing response time by several orders of magnitude. In complex multi-symptom scenarios, the LLM provided coherent, actionable insights faster than manual methods, underscoring its potential as a powerful assistant for real-time system health monitoring.

8. Conclusion

This paper presents a novel integration of the Model Context Protocol (MCP) with Elasticsearch to enable intelligent, LLM-driven system health monitoring. By addressing a longstanding challenge in observability;;the reliance on expert operators to interpret complex, interdependent system metrics, this work demonstrates how structured, schema-based communication can empower Large Language Models (LLMs) to act as autonomous diagnostic agents.

Through the development of a lightweight FastAPI-based MCP middleware, the system facilitates secure, auditable, and extensible interaction between an LLM and Elasticsearch's native APIs. The solution requires no modifications to Elasticsearch itself and can be deployed as a containerized microservice, making it scalable and easily adoptable in modern cloud-native environments.

Empirical evaluations revealed that the LLM, when guided by structured MCP inputs, achieved diagnostic accuracy on par with or exceeding that of experienced site reliability engineers. In scenarios of varying complexity, from single-node failures to multi-metric degradations, the LLM consistently provided actionable insights with sub-second response times. This significantly reduces mean time to resolution (MTTR) and supports a transition from reactive alerting to proactive system intelligence.

The primary novelty of this approach lies in combining a standardized LLM tool interface with a traditionally non-semantic telemetry system, effectively bridging the gap between passive monitoring and intelligent, language-driven diagnostics.

Future work will explore extending this architecture to support multi-source telemetry, including Prometheus [16] metrics, Grafana [16] dashboards, and log pipelines, along with automated remediation, anomaly-aware prompting [17], and broader interoperability with other observability platforms.

References

[1] Anthropic. (2024). Introducing the Model Context Protocol (MCP) https://www.anthropic.com/news/model-context-protocol
[2] Elastic. (2023). Elasticsearch API Reference. https://www.elastic.co/guide/en/elasticsearch/reference/index.html
[3] FastAPI. (2023). FastAPI Framework Documentation. https://fastapi.tiangolo.com/
[4] Lewis, P. et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS.
[5] Schick, T. et al. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. NeurIPS.
[6] Anthropic. (2024). Claude v3.5 Overview. https://www.anthropic.com
[7] Elastic. (2023). Elasticsearch API Reference. https://www.elastic.co/docs/api/doc/elasticsearch/operation/operation-cat-help
[8] Elastic. (2023). Elasticsearch Cluster, nodes and shards . https://www.elastic.co/docs/deploy-manage/distributed-architecture/clusters-nodes-shards
[9] Anthropic. (2024). Tooling Sandbox. https://www.anthropic.com/tools
[10] Docker. (2023). Docker Documentation. https://docs.docker.com/
[11] Kubernetes. (2023). Production-Grade Container Orchestration. https://kubernetes.io
[12] Google SRE. (2016). Site Reliability Engineering. O'Reilly Media.
[13] Elastic. (2023). Kibana. https://www.elastic.co/kibana
[14] OpenAI. (2023). Evaluating LLM Reasoning. https://openai.com/research
[15] Elastic. (2023). Securing Elasticsearch APIs. https://www.elastic.co/guide/en/elasticsearch/reference/current/security-api.html
[16] Grafana Labs. (2023). Observability Stack with Prometheus and Grafana. https://grafana.com
[17] ScienceLogic. (2023). Anomaly Detection Using AI/ML. https://sciencelogic.com