Here is a complete **Software Requirements Specification (SRS)** based on the problem statement provided.

Software Requirements Specification (SRS)

Project Name: OpsPulse AI (Real-Time Log Analysis & Automated Remediation)

Version: 1.0

Status: Draft

---

1. Introduction

1.1 Purpose

The purpose of this document is to define the functional and non-functional requirements for **OpsPulse AI**. This system is designed to ingest IT infrastructure logs in real-time, utilize Natural Language Processing (NLP) to classify log messages, detect statistical anomalies, and use Retrieval-Augmented Generation (RAG) to provide actionable remediation steps from a knowledge base of runbooks.

1.2 Scope

The system will:

- Ingest semi-structured log streams from various applications.
- Parse and normalize data (timestamps, error levels).
- Apply NLP models to classify log entries.
- Compute statistics over tumbling windows to identify anomalies (AIOps).
- Retrieve context-aware remediation guides via RAG.
- Alert administrators with both the error and the solution.

1.3 Definitions and Acronyms

- **AIOps:** Artificial Intelligence for IT Operations.
- **RAG (Retrieval-Augmented Generation):** A technique to optimize LLM output by referencing an authoritative knowledge base.
- **MTTR:** Mean Time To Resolution.
- **Runbook:** A compilation of routine procedures and operations that the system administrator or operator carries out.
- **Tumbling Window:** A fixed-size, non-overlapping time interval used for stream processing.

---

2. Overall Description

2.1 Product Perspective

OpsPulse AI acts as a middleware intelligence layer between raw infrastructure output (logs) and system administrators. It replaces passive logging dashboards with an active, predictive alerting system.

2.2 System Architecture

The system follows a stream-processing architecture:

1. **Source:** Applications/Servers generating logs.
2. **Ingestion Layer:** Kafka/Event Hub for message buffering.
3. **Processing Layer:** Stream processor (e.g., Flink/Spark) for parsing and windowing.
4. **Intelligence Layer:** NLP models for classification and Vector DB for RAG.
5. **Action Layer:** Alerting service and Dashboard.

2.3 User Classes and Characteristics

- **Site Reliability Engineers (SREs):** Primary users. They configure alerts, upload runbooks, and respond to incidents.
- **DevOps Engineers:** Integrate applications with the logging pipeline.
- **IT Managers:** View high-level MTTR statistics and system health.

2.4 Assumptions and Dependencies

- The system assumes access to a digitized repository of runbooks (PDF, Markdown, or Wiki).
- The system assumes logs contain at least a timestamp and a message payload.

---

3. System Features (Functional Requirements)

3.1 Log Ingestion and Parsing

Description: The system must accept high-velocity log data from multiple sources.

- **REQ-1.1:** The system shall support schema-flexible streaming to handle varying log formats (JSON, Syslog, Raw Text).
- **REQ-1.2:** The system shall parse logs to extract core metadata: `Timestamp`, `Source_IP`, `Log_Level` (INFO, WARN, ERROR), and `Message_Body`.
- **REQ-1.3:** The system must handle high throughput (minimum 10,000 logs/second) without data loss.

3.2 NLP Classification

Description: Raw text messages must be categorized to allow for statistical analysis.

- **REQ-2.1:** The system shall utilize an NLP model to classify unique log templates (e.g., grouping "Connection failed to IP 1.2.3.4" and "Connection failed to IP 5.6.7.8" as the same class: `DB_CONNECTION_FAIL`).
- **REQ-2.2:** The classification must occur in near real-time (< 2 seconds latency).

3.3 Anomaly Detection (AIOps Engine)

Description: The core logic to detect deviations from normal behavior.

- **REQ-3.1:** The system shall calculate statistics using **Tumbling Windows** (e.g., 1-minute or 5-minute distinct windows).
- **REQ-3.2:** The system shall detect **Spike Anomalies** (e.g., Error rate > 3 standard deviations from the moving average).
- **REQ-3.3:** The system shall detect **Drop Anomalies** (e.g., "Heartbeat" log frequency drops to zero).
- **REQ-3.4:** Thresholds for anomalies shall be dynamic/learnable or user-configurable.

3.4 Runbook Indexing and RAG Retrieval

Description: Connecting the problem to the solution.

- **REQ-4.1:** The system shall provide an interface for uploading technical runbooks and remediation guides (PDF, MD, Docx).
- **REQ-4.2:** The system shall index these documents into a Vector Database/Document Store.
- **REQ-4.3:** Upon detecting an anomaly, the RAG component shall query the Document Store using the log classification as the semantic query.
- **REQ-4.4:** The system shall extract the specific page or section relevant to the detected error.

3.5 Alerting and Notification

Description: delivering the actionable intelligence.

- **REQ-5.1:** The system shall generate an alert containing:
  - The Anomaly Description.
  - The Timestamp.
  - **The Recommended Remediation Step (retrieved from runbook).**
  - A link to the full runbook source.
- **REQ-5.2:** Alerts shall be deliverable via Email, Slack, or PagerDuty.

---

4. External Interface Requirements

4.1 User Interfaces

- **Dashboard:** A real-time visualization showing log volume, error rates per minute, and recent anomalies.
- **Runbook Manager:** A drag-and-drop interface for administrators to upload and manage remediation guides.

4.2 Hardware Interfaces

- The system is software-based but requires interaction with standard network interfaces for log collection (TCP/UDP ports).

4.3 Software Interfaces

- **Vector Database:** (e.g., Pinecone, Milvus) for storing runbook embeddings.
- **Message Broker:** (e.g., Kafka) for the log stream.
- **LLM API:** (e.g., OpenAI, Llama) for NLP processing and summarization.

---

5. Non-Functional Requirements

5.1 Performance

- **Latency:** The time from log generation to anomaly detection alerting must be under 60 seconds.
- **Scalability:** The architecture must support horizontal scaling of stream consumers to handle log spikes.

5.2 Reliability

- **Availability:** 99.9% uptime.
- **Fault Tolerance:** If the NLP service fails, raw logs must still be archived even if classification is temporarily unavailable.

5.3 Security

- **Data Privacy:** Logs often contain sensitive PII. The system must support masking of sensitive fields (e.g., credit card numbers, passwords) during the parsing phase.
- **Access Control:** Only authorized SREs can modify runbooks or alert thresholds (RBAC).

---

6. Data Flow Diagram (Conceptual)

1. **Input:** Raw Log Stream -> **Parser** (Extracts Time/Level).
2. **Processing:** Parsed Log -> **NLP Model** (Assigns Class ID).
3. **Aggregation:** Class ID -> **Windowing Function** (Counts per minute).
4. **Decision:** Count vs Threshold -> **Anomaly Detector**.
   - *If Normal:* Store metric, do nothing.
   - *If Anomaly:* Trigger **RAG Retriever**.
5. **Retrieval:** Anomaly Context -> **Vector DB** -> Retrieve Runbook_Snippet.
6. **Output:** Alert + Runbook_Snippet -> **Admin**.

---

7. Future Enhancements

- **Auto-Remediation:** Allow the system to execute the runbook commands (scripts) automatically for low-risk anomalies (Self-Healing).
- **Root Cause Analysis:** Correlate logs from different services to find the cascading source of the error.

Would you like me to create a Python simulation code for the "Tumbling Window" statistical analysis part of this SRS?

Software Requirements Specification (SRS)

Project Name: OpsPulse AI (Real-Time Log Analysis & Automated Remediation)

Version: 1.0

Status: Draft

---

1. Introduction

1.1 Purpose

The purpose of this document is to define the functional and non-functional requirements for **OpsPulse AI**. This system is designed to ingest IT infrastructure logs in real-time, utilize Natural Language Processing (NLP) to classify log messages, detect statistical anomalies, and use Retrieval-Augmented Generation (RAG) to provide actionable remediation steps from a knowledge base of runbooks.

1.2 Scope

The system will:

- Ingest semi-structured log streams from various applications.
- Parse and normalize data (timestamps, error levels).
- Apply NLP models to classify log entries.
- Compute statistics over tumbling windows to identify anomalies (AIOps).
- Retrieve context-aware remediation guides via RAG.
- Alert administrators with both the error and the solution.

1.3 Definitions and Acronyms

- **AIOps:** Artificial Intelligence for IT Operations.
- **RAG (Retrieval-Augmented Generation):** A technique to optimize LLM output by referencing an authoritative knowledge base.
- **MTTR:** Mean Time To Resolution.
- **Runbook:** A compilation of routine procedures and operations that the system administrator or operator carries out.
- **Tumbling Window:** A fixed-size, non-overlapping time interval used for stream processing.

---

2. Overall Description

2.1 Product Perspective

OpsPulse AI acts as a middleware intelligence layer between raw infrastructure output (logs) and system administrators. It replaces passive logging dashboards with an active, predictive alerting system.

2.2 System Architecture

The system follows a stream-processing architecture:

1. **Source:** Applications/Servers generating logs.
2. **Ingestion Layer:** Kafka/Event Hub for message buffering.
3. **Processing Layer:** Stream processor (e.g., Flink/Spark) for parsing and windowing.
4. **Intelligence Layer:** NLP models for classification and Vector DB for RAG.
5. **Action Layer:** Alerting service and Dashboard.

2.3 User Classes and Characteristics

- **Site Reliability Engineers (SREs):** Primary users. They configure alerts, upload runbooks, and respond to incidents.
- **DevOps Engineers:** Integrate applications with the logging pipeline.
- **IT Managers:** View high-level MTTR statistics and system health.

2.4 Assumptions and Dependencies

- The system assumes access to a digitized repository of runbooks (PDF, Markdown, or Wiki).
- The system assumes logs contain at least a timestamp and a message payload.

---

3. System Features (Functional Requirements)

3.1 Log Ingestion and Parsing

Description: The system must accept high-velocity log data from multiple sources.

- **REQ-1.1:** The system shall support schema-flexible streaming to handle varying log formats (JSON, Syslog, Raw Text).
- **REQ-1.2:** The system shall parse logs to extract core metadata: Timestamp, Source_IP, Log_Level (INFO, WARN, ERROR), and Message_Body.
- **REQ-1.3:** The system must handle high throughput (minimum 10,000 logs/second) without data loss.

3.2 NLP Classification

Description: Raw text messages must be categorized to allow for statistical analysis.

- **REQ-2.1:** The system shall utilize an NLP model to classify unique log templates (e.g., grouping "Connection failed to IP 1.2.3.4" and "Connection failed to IP 5.6.7.8" as the same class: DB_CONNECTION_FAIL).
- **REQ-2.2:** The classification must occur in near real-time (< 2 seconds latency).

3.3 Anomaly Detection (AIOps Engine)

Description: The core logic to detect deviations from normal behavior.

- **REQ-3.1:** The system shall calculate statistics using **Tumbling Windows** (e.g., 1-minute or 5-minute distinct windows).
- **REQ-3.2:** The system shall detect **Spike Anomalies** (e.g., Error rate > 3 standard deviations from the moving average).
- **REQ-3.3:** The system shall detect **Drop Anomalies** (e.g., "Heartbeat" log frequency drops to zero).
- **REQ-3.4:** Thresholds for anomalies shall be dynamic/learnable or user-configurable.

3.4 Runbook Indexing and RAG Retrieval

Description: Connecting the problem to the solution.

- **REQ-4.1:** The system shall provide an interface for uploading technical runbooks and remediation guides (PDF, MD, Docx).
- **REQ-4.2:** The system shall index these documents into a Vector Database/Document Store.
- **REQ-4.3:** Upon detecting an anomaly, the RAG component shall query the Document Store using the log classification as the semantic query.
- **REQ-4.4:** The system shall extract the specific page or section relevant to the detected error.

3.5 Alerting and Notification

Description: delivering the actionable intelligence.

- **REQ-5.1:** The system shall generate an alert containing:
  - The Anomaly Description.
  - The Timestamp.
  - **The Recommended Remediation Step (retrieved from runbook).**
  - A link to the full runbook source.
- **REQ-5.2:** Alerts shall be deliverable via Email, Slack, or PagerDuty.

---

4. External Interface Requirements

4.1 User Interfaces

- **Dashboard:** A real-time visualization showing log volume, error rates per minute, and recent anomalies.
- **Runbook Manager:** A drag-and-drop interface for administrators to upload and manage remediation guides.

4.2 Hardware Interfaces

- The system is software-based but requires interaction with standard network interfaces for log collection (TCP/UDP ports).

4.3 Software Interfaces

- **Vector Database:** (e.g., Pinecone, Milvus) for storing runbook embeddings.
- **Message Broker:** (e.g., Kafka) for the log stream.
- **LLM API:** (e.g., OpenAI, Llama) for NLP processing and summarization.

---

5. Non-Functional Requirements

5.1 Performance

- **Latency:** The time from log generation to anomaly detection alerting must be under 60 seconds.
- **Scalability:** The architecture must support horizontal scaling of stream consumers to handle log spikes.

5.2 Reliability

- **Availability:** 99.9% uptime.
- **Fault Tolerance:** If the NLP service fails, raw logs must still be archived even if classification is temporarily unavailable.

5.3 Security

- **Data Privacy:** Logs often contain sensitive PII. The system must support masking of sensitive fields (e.g., credit card numbers, passwords) during the parsing phase.
- **Access Control:** Only authorized SREs can modify runbooks or alert thresholds (RBAC).

---

6. Data Flow Diagram (Conceptual)

1. **Input:** Raw Log Stream -> **Parser** (Extracts Time/Level).
2. **Processing:** Parsed Log -> **NLP Model** (Assigns Class ID).
3. **Aggregation:** Class ID -> **Windowing Function** (Counts per minute).
4. **Decision:** Count vs Threshold -> **Anomaly Detector**.
   - *If Normal:* Store metric, do nothing.
   - *If Anomaly:* Trigger **RAG Retriever**.
5. **Retrieval:** Anomaly Context -> **Vector DB** -> Retrieve Runbook_Snippet.
6. **Output:** Alert + Runbook_Snippet -> **Admin**.

---

7. Future Enhancements

- **Auto-Remediation:** Allow the system to execute the runbook commands (scripts) automatically for low-risk anomalies (Self-Healing).
- **Root Cause Analysis:** Correlate logs from different services to find the cascading source of the error.

Would you like me to create a Python simulation code for the "Tumbling Window" statistical analysis part of this SRS?