**Title:** AI-Powered Incident Resolution Agent Architecture Document

## **1. Introduction**

### **1.1 Purpose**

This document outlines the architecture of an AI-powered incident resolution agent. The system leverages Python and React to provide a seamless user experience while integrating with Elastic APM, Prometheus, and telemetry data sources. AI-driven analysis and resolution mechanisms enhance issue detection and response based on knowledge base articles and historical incident data.

### **1.2 Scope**

* Fetch telemetry data from Elastic APM, Prometheus, and other monitoring sources.
* Process and analyze data using AI models.
* Identify and resolve issues based on knowledge base articles and past incident reports.
* Provide a user-friendly interface in React.
* Ingest incident reports via email and store them in the database for analysis.

## **2. System Architecture**

### **2.1 High-Level Design**

The architecture consists of the following core components:

* **Frontend:** Built using React for an interactive user experience.
* **Backend:** Developed in Python (FastAPI/Django) for API handling and business logic.
* **AI Engine:** NLP and ML models for analyzing telemetry data and issue resolution.
* **Data Ingestion Layer:** Collects and normalizes data from Elastic APM, Prometheus, telemetry sources, and email-based incident reports.
* **Knowledge Base & Incident Repository:** Stores articles and historical incident data for AI-driven analysis.
* **Monitoring & Logging:** Ensures continuous tracking of system performance and anomalies.

### **2.2 Component Breakdown**

#### **2.2.1 Frontend (React)**

* Displays real-time monitoring insights.
* Provides interactive dashboards for issue visualization.
* Enables user interactions with AI-based recommendations.

#### **2.2.2 Backend (Python – FastAPI/Django)**

* Handles API requests from the frontend.
* Fetches data from monitoring sources and knowledge repositories.
* Invokes AI-based issue resolution mechanisms.
* Logs actions for audit and tracking.
* Parses incoming email reports and stores incident data in the database.

#### **2.2.3 AI Engine**

* Utilizes machine learning (ML) and natural language processing (NLP) to analyze data.
* Correlates telemetry and incident data to predict and resolve issues.
* Recommends solutions based on past resolutions and documentation.

#### **2.2.4 Data Ingestion Layer**

* Uses Python-based collectors to pull data from:
  + **Elastic APM** (application performance metrics)
  + **Prometheus** (infrastructure monitoring)
  + **Custom telemetry sources**
  + **Email reports** (incident data ingestion from structured emails)
* Normalizes and stores data for analysis.

#### **2.2.5 Knowledge Base & Incident Repository**

* Stores and retrieves past incidents and resolutions.
* Enhances AI recommendations by leveraging existing documentation.
* Indexed for quick retrieval using Elasticsearch or a relational database.
* Stores parsed incident data from email reports.

## **3. Data Flow**

1. The frontend (React) sends a request to the backend (Python) for monitoring insights.
2. The backend retrieves data from Elastic APM, Prometheus, and telemetry sources.
3. The system processes incoming email-based incident reports and stores them in the database.
4. AI Engine processes data, identifying potential issues using ML models.
5. The system fetches knowledge base articles and past incidents for resolution suggestions.
6. The AI engine provides recommendations to the frontend.
7. The user can either apply the suggested resolution or manually override it.
8. Logs and monitoring dashboards update accordingly.

**5. Technology Stack**

|  |  |
| --- | --- |
| **Component** | **Technology** |
| Frontend | React |
| Backend | Python (FastAPI/Django) |
| AI Engine | TensorFlow, PyTorch, NLP libraries |
| Data Ingestion | Elastic APM, Prometheus, OpenTelemetry, Email Parsing |
| Database | PostgreSQL, Elasticsearch |
| Monitoring & Logging | Telemetry, Prometheus |

## **6. Security & Scalability Considerations**

### **6.1 Security**

* Authentication via OAuth2/JWT.
* Secure API endpoints with rate limiting.
* Data encryption in transit and at rest.
* Secure email parsing to prevent phishing and injection attacks.

### **6.2 Scalability**

* Use Kubernetes for microservices deployment.
* Implement load balancing with Nginx/HAProxy.
* Optimize AI processing with distributed computing.
* Ensure efficient email ingestion pipelines for high-volume incident reporting.

## **7. Conclusion**

This AI-powered agent automates incident resolution using a combination of real-time monitoring, AI analysis, and historical knowledge retrieval. The modular architecture ensures scalability, security, and efficiency in handling system anomalies and performance issues. Additionally, integrating email-based incident reports enhances the system’s ability to track and analyze issues in real time.