AI Anomaly Detection System - Architecture Document

Version: 1.0
Last Updated: March 12, 2025

1. Introduction

1.1 Purpose

This document outlines the architecture of the AI Anomaly Detection System, detailing its components, interactions, and implementation strategy. The system detects anomalies in real-time data streams and adapts through a feedback-based retraining mechanism.

1.2 Scope

- **Real-time anomaly detection** using AI (Autoencoder)
- **Live visualization** of detected anomalies
- **User feedback loop** to refine AI predictions
- **Cloud-based deployment** with scalable components

1.3 Stakeholders

- **Developers** (implementing and maintaining the system)
- **Data Scientists** (fine-tuning AI models)
- **End-users** (monitoring anomalies and providing feedback)
- **DevOps Engineers** (deploying and maintaining infrastructure)

---

2. Architectural Drivers

2.1 Business Goals

- Provide real-time anomaly detection for streaming data
- Improve detection accuracy through continuous learning
- Ensure a scalable, cloud-native architecture

2.2 Key Functional Requirements

- Process and analyze incoming data streams
- Detect anomalies using AI
- Display anomalies in real time
- Collect user feedback for model retraining

2.3 Key Non-Functional Requirements

- **Performance:** Process data with minimal latency
- **Scalability:** Handle increasing data volumes
- **Security:** Secure API endpoints and data storage
- **Resilience:** Ensure system reliability under load

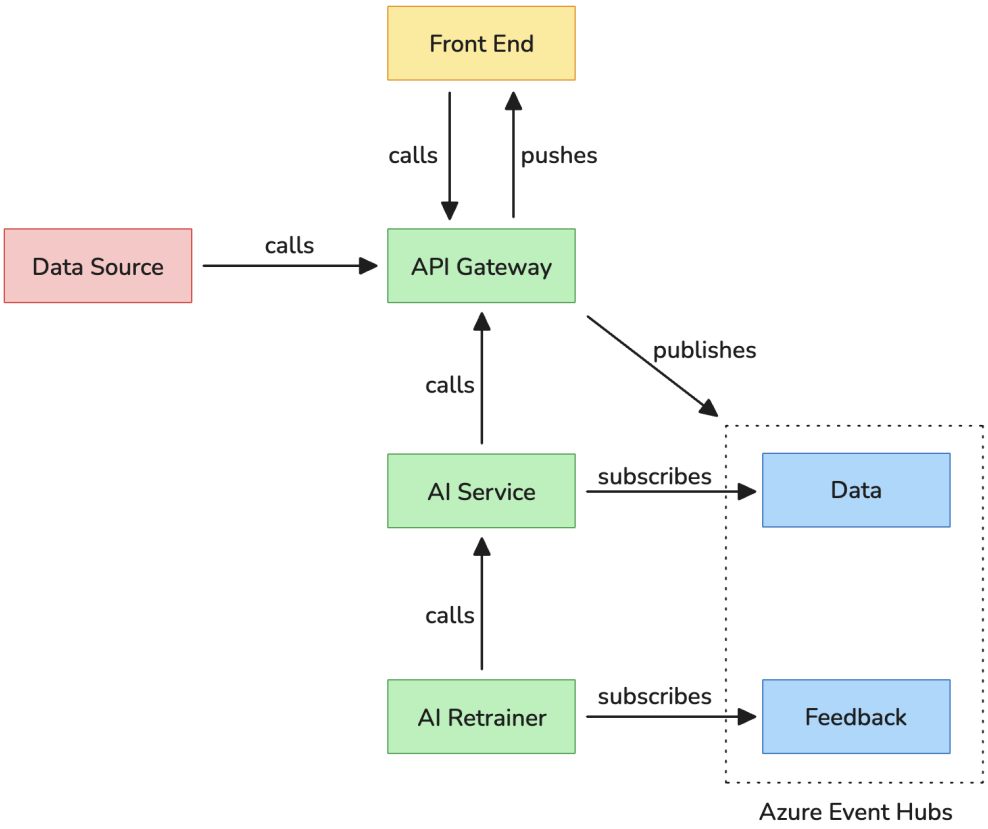
2.4 Constraints

- Cloud-native (Azure-based deployment)
- Uses FastAPI for AI processing
- Data ingestion via Azure Event Hubs

---

3. System Overview

3.1 High-Level Architecture Diagram



3.2 Tech Stack

Component	Technology
Front End	React, TypeScript, ReCharts
API Gateway	ASP.NET Core (WebAPI, SignalR hub)
Data Streaming	WebAPI, Azure Event Hub
AI Processing	Python (FastAPI, Autoencoder, PyTorch)

---

4. Architectural Decisions

4.1 Technology Choices

- **SignalR** for real-time frontend updates.
- **Azure Event Hub** for real-time data streaming to enable scalable ingestion.
- **ASP.NET Core WebAPI** for strong integration with Azure and overall robustness in cloud-native applications.
- **FastAPI** for AI processing due to Python's prevalence in machine learning.
- **Autoencoder Model** for unsupervised anomaly detection.

4.2 Trade-offs

- Using **SignalR** for real-time updates simplifies frontend communication but increases dependency on WebSockets.
- **Azure-based deployment** ensures scalability but may introduce cloud service costs.

---

5. System Components

5.1 Front End

- Displays detected anomalies in real-time using **ReCharts**.
- Connects via **SignalR** to receive updates.
- Provides feedback controls for marking anomalies as valid/invalid.

5.2 API Gateway

- Accepts new data and feedback via **REST API**.
- Pushes data and feedback to **Azure Event Hubs**.
- Routes AI detection results to the frontend in real-time using **SignalR**.

5.3 AI Service

- Consumes data from the event hub.
- Runs anomaly detection using an **Autoencoder** model.
- Returns detections to the frontend via the API Gateway.
- Accepts model updates via a FastAPI endpoint.

5.4 AI Model Retrainer

- Listens to user feedback in the event hub.
- Retrains the model when confidence drops.
- Deploys updated models to the AI Service via its FastAPI endpoint.

---

6. Deployment & Infrastructure

- **Hosting:** Azure (App Services, Event Hub, Key Vault).
- **CI/CD:** GitHub Actions for automated deployment.
- **Security:** JWT authentication for API and SignalR.

---

7. Data Model

7.1 Event Data Structure

```
{
  “Id”: “9b1542f6-45c3-4641-a77d-eb8f43e83a6d”,
  "timestamp": "2025-03-12T12:34:56Z",
  "sensor_id": "sensor_001",
  "value": 47.3
}
```

7.2 Anomaly Detection Output

```
{
  “data”: { }
  "anomaly_score": 0.92,
  "is_anomalous": true
}
```

---

8. API & Integration

8.1 API Gateway - Key API Endpoints

Endpoint	Method	Purpose
`/data`	POST	Receive new sensor data
`/feedback`	POST	Accept user feedback
`/anomaly`	POST	Send detected anomalies

8.2 AI Service - Key API Endpoints

Endpoint	Method	Purpose
`/update`	POST	Upload a new AI model

8.3 Event Streaming Topics

- **Data Queue**: Raw input data
- **Feedback Queue**: User feedback for retraining
- **Anomaly SignalR Hub**: Detected anomalies pushed to the front end

---

9. Operational Considerations

9.1 Monitoring & Logging

- **Azure Monitor** for system health
- **Application Insights** for performance tracking
- **Logging** of anomaly detection results

9.2 Error Handling

- **Retry logic** in event consumers
- **Dead-letter queues** for failed messages

9.3 Performance Considerations

- **Batch processing** for incoming data
- **Load balancing** for AI processing nodes

---

10. Future Considerations

- **Enhancements:** Fine-tune the anomaly detection model.
- **Scalability:** Support multiple AI models for different use cases.
- **Data Persistence:** Retain input data and anomaly detections for front-end browsing and post-hoc feedback.